Hitool for ARM® User's Manual

Hitool SYSTEM INC.

Http://www.hitoolsys.com

Doc. No. 003-000041
Fourth Edition
November 26, 2002

Trademark Acknowledgments

ARM is a registered trademark of ARM Ltd.
Microsoft, MS and Windows are registered trademarks of Microsoft Corporation.
Intel is a registered trademark of Intel Corporation.

Document Number 003-000041 (Fourth Edition)
©2001 Hitool INTERNATIONAL INC.
All Rights Reserved.
Printed in Taiwan 11/2002

The contents of this manual are subject to change without notice. Hitool International Inc., assumes no responsibility for errors that may appear in this manual. Hitool makes no commitment to update, nor to keep current, the information contained in this manual. Nothing herein shall be construed as a recommendation to use any product in violation of existing patents or other rights of third parties. No part of this manual may be reproduced or transmitted in any form or by any other means without prior written permission from Hitool.

Hitool is copyrighted by Hitool International Inc., and is furnished under a license or nondisclosure agreement, and may be used or copied only in accordance with the terms of the agreement.

USER MAY NOT USE, COPY, MODIFY, OR TRANSFER Hitool, OR MAKE ANY COPY THEREOF, EXCEPT AS EXPRESSED IN THE AGREEMENT.

HITOOL SYSTEM INC.

Web : www.hitoolsys.com
E-mail : sales@hitoolsys.com

Contents

1. Introduction 1

About This Manual 1
Hitool--The Next Generation of Point and Click Support Tools 1
Software Debugger 1
Setting a New Standard for Intuitive 2
Customizable Toolbars 2
Intuitive Debugging Windows 3
Easy Project Configuration 3
All New Graphical Symbols 3
Fast and Easy Source Browsing 4
Powerful and Easy Data Watching 4
Quick and Easy Display of Calls 5
Flexible Memory Display 6
Easy Viewing of Registers 7
Fast, Robust Target Control 9
Project-Wide 'Grep' 10
Product Warranty 10
Related Publications 10
Registering 10

2. Getting Started 11

System Requirements 11
Windows Requirements 11
Installing Hitool for ARM 11
Hitool for Windows 9x/NT 11
About NT driver 15
About Dongle 15
Registering 16
Checking Host Configuration 17
Configuring Communications 17
Debugging your own project 21

Downloadable Formats 22
Hitool IDE 22
Project Management 22
IDE 23
Load Options 24
Load Type 25
Downloading the Binary Image and Reading Symbols 25
Project Parameters 28
Troubleshooting Tips 29

3. Configuring a Project and Building Target 31

Building Target 31
Configuring Compiler Path 31
Creating a new project 32
Configuring Attributes of the Project 36
Adding Source Files 37
Building Project 42
Overview of the Project Settings Dialog Box 43
Using the Project Settings dialog box 44
Configuring General Target Options 45
Configuring Assembler Settings 47
Configuring Compiler Settings 50
Configuring Library and Linking Settings 57
Using the Equivalent Command Line text box 66

4. Viewing & Editing Source Code 67

Viewing Source Code 67
The Source Window 67
Source Window Pop-Up Menus 69
Source Display Format 72
Source Paths 72
Specifying Source Path 73
Editing Source Code 73
Undoing changes 73
Searching and Replacing Text 74
Closing the Source Window 76

5. Controlling Execution 77

Controlling Execution 77
Debug Toolbar 77
Debug Menu 77

6. Displaying Data, Memories, Registers 81

Displaying Data 81
Datatip 81
Inspect 82
Stack Window 83
Displaying Memories, Registers 84
Display Memories 84
Display Registers 86
Watch Window 93
Opening the Watch Window 93
Placing Data into the Watch Window 94
Placing Registers, Memories into the Watch Window 96
Viewing Data, Registers, Memories in the Watch Window 99
Changing Data 100
Changing Registers, Memories 100

7. Breakpoints&Watchpoints 103

Breakpoints&Watchpoints 103
Software Breakpoints 103
Watchpoints 104
Setting Breakpoints 105
Source Window 105
Setting Breakpoint through Breakpoint dialog box 106
Setting Breakpoint through Breakpoints Window 107
Setting Conditional Software Breakpoint through Breakpoint dialog box 107
Breakpoint dialog box 109
Breakpoints Window 110
Project Window Symbols Pane 111
Stacks Window 112
Setting Watchpoints 113
Setting Watchpoints for Angel series 113
Setting Watchpoint through Source Window 113
Setting Watchpoint through Watchpoint dialog box 114
Setting Watchpoints for Wiggler series 115
Output Window Command Pane 117
Configuring Exception Trapping Settings 117

8. Project Window 119

Project Window Overview 119
Working with Project 119
Working with Symbols 120

Add or Remove Symbol Table 121
Drag and Drop Capabilities 123
Modules 123
Functions 124
Variables 125
Viewing Class 126
Functions 127
Type Name 128
Closing the Project Window 128

9. Memory Window **131**

Memory Window 131
Editing Memories 132
Memory Action 133
Upload/Download Memory 139
Viewing Memory 140
Closing the Memory Window 140

10. ThreadX Window **141**

Hitool ThreadX Window 141
System Pane 141
Thread Pane 142
Queue Pane 145
Semaphore Pane 147
Event Flags Pane 148
Timer Pane 150
Block Pool Pane 151
Byte Pool Pane 153

11. Miscellaneous Tools **157**

Convert 157
Programming Flash with the ARM Utilities Panel GUI 158
Programming Flash - Step by Step 159
Logging and Running Include 160
Logging On/Off 161
Writing Include 161
Running Include 162
Writing and Using Macro 162
Customizable Toolbars 163
Moving Toolbars 163
Customizing Toolbars 164

Action Record 165
Recording 166
Playback 166

12. Command Reference 167

Shell Command 167
Syntax Notational Conventions 168
Commands and System Variables Grouped by Function 168
$variable 170
$variable [=value] 170
ACcess 170
ACcess [-B | -Hw | -W] 170
ALias 170
ALias [original_name new_name] 170
Assemble 171
Assemble [adr] 171
Breakpoint 171
BReakpoint [adr] 171
BYte 172
BYte adr datum1 [..datum32] 172
CLear 172
CLear {All | adr | HW0 | HW1} 172
COMpare 173
COMpare adr1 {adr2 | Length length} adr3 [-B | -Hw | -W] 173
Copy 174
Copy adr1 {adr2 | Length length} adr3 [-B | -Hw | -W] 174
CReate 174
CReate #symbol adr 174
DElete 175
DElete #symbol | $variable | All 175
Disassemble 175
Disassemble [adr1 [adr2 | Length length]] 175
Fill 176
Fill adr1 {adr2 | Length length} {“string” | data..} [-B | -Hw | -W] 176
Go 177
Go [[Run] [From adr1] | [From adr1] [Till adr2] [Monitor timer]] 177
HAlt 177
HAlt 177
Help 177
Help [command] 177
Include 178
Include filename 178

Jump 178
Jump adr 178
LOad 178
Load filename [Symbols | NOSymbols] [Code | NOCode] [Warn | NOWarn] [S-R | BIN addr | DIS addr | DISA | DISC addr | DISAC | EXEC] 178
LOG 180
LOG {filename [Command] [Append]} | {ON | Off | CLear} 180
LONg 180
LONg adr data 1 [..data 8] 180
Memory 180
Memory adr1 [adr2 | Length length] [-B | -Hw | -W] 180
QUEry 181
QUEry #symbol | hex_adr 181
Register 182
Register [{reg_id} [data]] 182
RESet 182
RESet [adr] 182
RESTart 182
RESTart 182
SAvedisasm 183
SAvedisasm adr1 {adr2 | Length length} file 183
SRecall 183
SRecall filename 183
SSave 184
SSave filename 184
Step 184
Step [Over][count | FOrever] 184
SYmbol 184
SYmbol [Global | Module | #module | Lines] 184
TEst 185
TEst addr1 {adr2 | Length length} [Mem | Data | Addr] [-B | -Hw | -W] 185
Upload 186
Upload adr1 { adr2 | Length length } filename [S-R | BIN | DIS | DISA | DISC | DISAC] 186
WAtch 187
WAtch "C-Expression" 187
Watchpoint 187
Watchpoint [ADdr addr] [AMask mask] [DAta data] [DMask mask] [-B | -W] [Fetch | Access] [Read | Write] [HW0 | HW1] 187
Word 188
Word adr data1 [.. data16] 188

13. Semihosting And Misc **189**

Overview....189
Semihosting Handle....189
RDILog pane....191
Target Info dialog box....192
Comms Channel Viewer....192
Console view....193

A. Compiler Configuration and JTAG Port **195**

Compiler Configuration....195
ARM Compiler Configuration....195
ADS Compiler Configuration....197
JTAG port....199

Contents **iii**

1 Introduction

About This Manual

This manual describes all the details about Hitool source level debugger for ARM.

> ***NOTE***
>
> *The ARM emulation board is referred to as the "emulator" throughout this manual.*

Hitool--The Next Generation of Point and Click Support Tools

Hitool is a "convenience driven, point and click" support tool. Combining state-of-the-art debugging facilities with the intuitive simplicity of a Windows-based environment, each major function contains a "point and click" window or tool to make debugging fast, flexible, and direct.

Now, Hitool makes it even easier to access multiple functions and windows from not only one main source window, but from multiple access points. Hitool runs under Windows 95 above and Windows NT 4.0 above.

The power and functionality of Hitool provide the ideal environment for developing with these emulation and debug vehicles.

Software Debugger

Hitool is a high performance on-chip C / C++ debugger. Hitool provides a high-speed parallel connection between a host debugger and target microprocessor. With lightning-quick stepping and the ability to reduce download time from minutes to seconds, you'll spend less time waiting and more time actually debugging your applications.

The cutting edge Graphical User Interface of Hitool and the raw power supplied by Hitool combine to give you the perfect environment for developing state-of-the-art software applications. In addition to being a full-featured software debugger, Hitool offers a comprehensive set of advanced features to minimize your time-to-market. With functionality that extends beyond debugging, you get a tool that is also useful for bringing up hardware, initializing on-board chip peripherals, and burning code into Flash.

Designed for Windows 9x / NT, Hitool delivers a solution powerful enough to uproot and destroy even the nastiest of bugs quickly and efficiently, yet is affordable enough for every developer on your team to have.

Setting a New Standard for Intuitive

By using a Windows 95 graphical user interface, Hitool helps make your embedded development project move faster and easier, helping you beat your competitors to market, hands down. Time-saving features include:

- Customizable Toolbars
- Detachable Menus - Create Your Own Toolbars
- Drag and Drop Information From One Window to Another
- Right Mouse-click / Pop-up Menu Support, with tool tips for most icons

Customizable Toolbars

Menus from toolbar are detachable so that you can create additional customized toolbars. A powerful Hitool macro language makes it possible to alter the pre-programmed icons to support different commands or to include further functions beyond a single task. Alternatively, icons can be entirely re-programmed and re-labeled to support other desired features.

In addition, when you place your mouse over an icon on the toolbar, you'll see a tool tip appear for that icon.

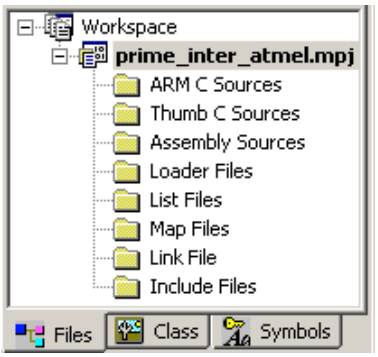
The following sections will introduce some of the enhanced timesaving features of Hitool for Windows 9x and Windows NT. For more detailed information on any of these topics, refer to the chapters which follow in this manual.

Intuitive Debugging Windows

Hitool is a full Windows 9x / Windows NT MDI (Multiple Document Interface) application. This means that you can have any number of its child windows open simultaneously. The sections below provide an overview of the capabilities of these windows, and how you can use them to expose complex bugs, swiftly and easily.

Easy Project Configuration

Project Window can be opened automatically after you launch Hitool. You can display or hide Project Window by adding or removing the check mark beside Project Window from the View menu.

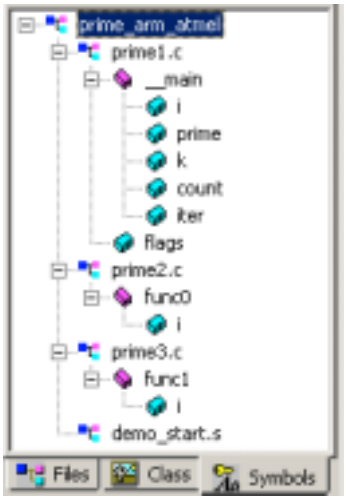


Right click on any setting within the Project Window, and you have instant access to key project settings.

All New Graphical Symbols

The Project Window Symbols pane allows you to view a full graphical display of your project symbols. The Symbols pane offers convenient drag and drop capabilities, as well as right-click menus.

For added convenience, you can expand and contract Modules, Functions, Locals, and Globals with just a click of the mouse. Simply click on a module or a function, variables and you will see the module selected, type, address, attribute and size.



Fast and Easy Source Browsing

The Source Window is the primary window for all views of your application.

Right-click anywhere on the Source Window, and a pop-up menu appears, enabling you to perform a variety of operations easily and efficiently to easily control the execution and monitoring of variables inside the Source Window.

Simple Viewing of Modules and Functions

When you read symbols into Hitool, the software builds an internal table of all of the modules and functions that you have compiled (and / or assembled) and linked to form your application.

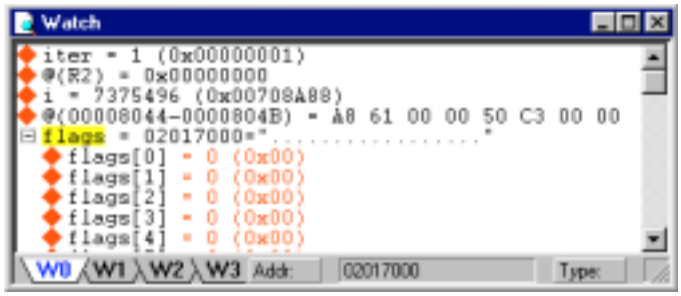
Powerful and Easy Data Watching

Undoubtedly, one of the primary purposes of a source-level debugger is to evaluate and monitor high-level variables at various points in your application program. Hitool makes it easy for you to accomplish this task using the Hitool GUI – dramatically improved in the form of a new, high-powered, outline-based Watch Window. Further, Hitool supports both simple and complex data type, such as arrays and class data.

The Watch Window: Continuous Data Monitoring

The Watch Window provides an outline format for watching data. If there is a variable that you want to continuously monitor, simply add the variable into the Watch Window multiple Watch tabs.

The Watch Window consists of four separate tabs. The tabs enable you to evaluate local variables on the current Call Stack and global variables, memory content and registers value. Every time the target stops, the Watch Window is updated with the data's most current value.

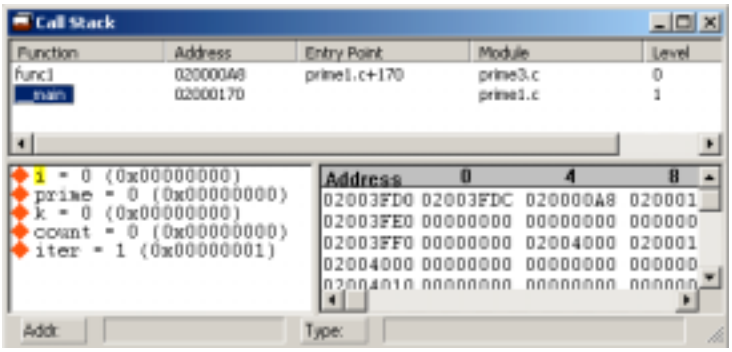


Right-click functionality allows you to click anywhere in the Watch Window and a pop-up menu appears, enabling you to perform a variety of operations.

If you wish to see the value of a variable without permanently stopping the target, just highlight the variable in the Source Window, choose Inspect from the pop-up menu. This will add the variable into the Inspect Window which is always on the top.

Quick and Easy Display of Calls

The Stack Window



Stack Window

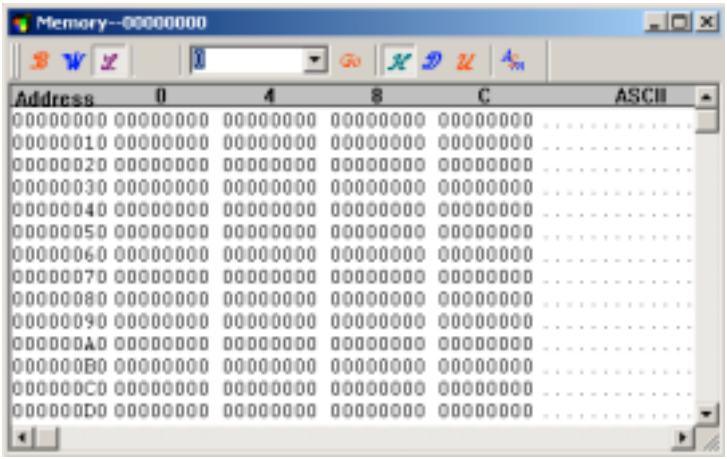
The Call Stack Window provides a quick and easy display of all of the functions active on the calling chain and the memory content corresponding to the function stack pointer of highlighted function.

The Call Stack Window can also be used with the Watch Window's tab to view local variables of a function on the calling chain.

Flexible Memory Display

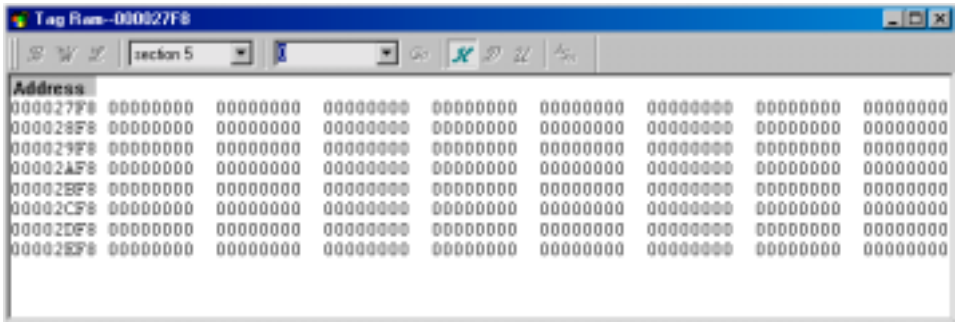
The Memory Window

The Memory Window provides a powerful format to display target memory in Hex, Decimal, ASCII, and binary formats -- you select the formats depending on your needs. Continuously scrollable, the window allows you to view, compare, fill, test and copy blocks of memory starting at any address you choose.



The PA TAG RAM Window

920T implements PA TAG RAM to perform write-back from Data Cache. Scan Chain 4 can view it, so Hitool can provide an UI to display it to make user aware. This view is read only.



Easy Viewing of Registers

The Register Window

The Register Window allows easy viewing and modification of target registers.

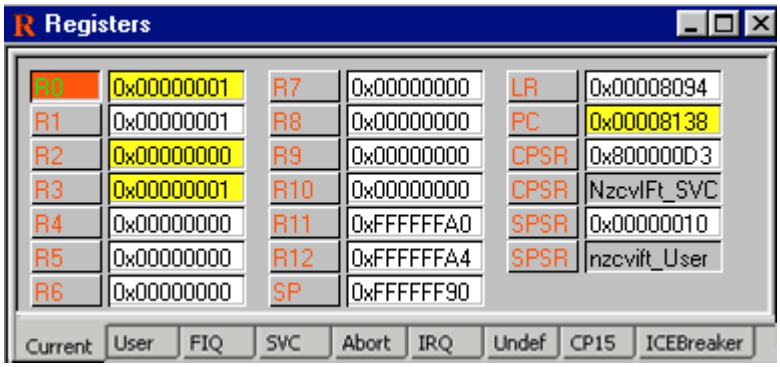
ICEBreaker pane will be added while using Wiggler, Raven, PowerProbe, and PowerProbe+ Protocol, and registers in this pane are read-olny. With the Vector Catch register in the ICEBreaker pane, the debugger can set the corresponding bit to catch the vector instead of settings a software breakpoint in the vector address.

Using Angel or Jeeni Protocol, some Coprocessor register pane will be added. Debugger will auto detect how many Co-processors is available after connecting successfully. For example, for Samsung KS32C50100 and Atmel AT91, CP0 (ICEBreaker in fact) is available for Jeeni protocol, but no coprocessor is available for Angel protocol.

An outline control updates only what you want updated. For your convenience, we have also included drag and drop capabilities in the Watch Windows.

Changing set values is easy -- just double click on the register buttons then fill in the new value directly.

Register Window displays or changes the target processor general register value.



Register Window

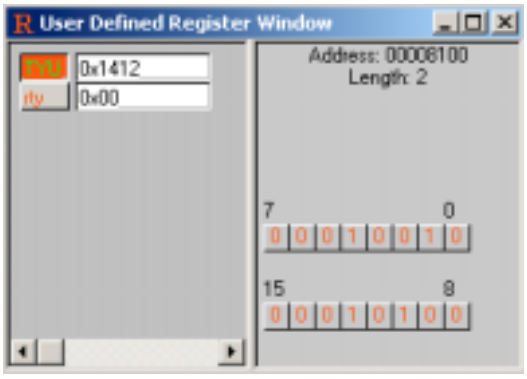
The User Defined Register Window

The User Defined Register Window allows easy viewing and modification of user defined registers.

The User Defined Register Window consists of two panels: the left one displays all the user defined registers similar to the Register Window; the right one displays the information of the selected user defined register and its each bit if it is bit-able.

An outline control updates only what you want updated.

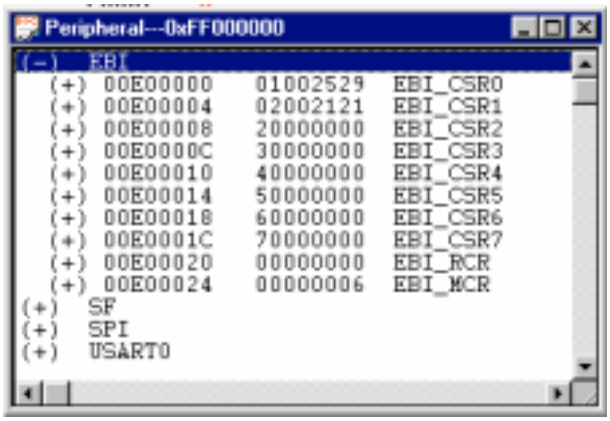
Changing set values is easy -- just double click on the register buttons then fill in the new value directly.



User Defined Register Window

The Peripheral Window

Peripheral Window displays memory-mapped control registers.



Fast, Robust Target Control

Hitool uses a compiled VBScript which offers dramatic target control improvement. In addition, Hitool delivers tuned-up stepping while still maintaining the same pre-emptive architecture. With Up-Level Breakpoint and Stay at High-Level, two features, you get even *more* speed for target control.

In addition, you can perform a number of target control actions by simply pressing a function key:

- Start / Stop execution (Go, Goto Cursor, Free Run, Halt)
- Single-step at the source level either into, over, or out of functions
- Single-step at the instruction level
- Reset the target system

Flexible and Intuitive Breakpoint and Watchpoint Management

With Hitool, you can easily set both breakpoint and watchpoint from Source, Mixed Source and Disassembly window. By clicking on anywhere within the Source, Mixed Source and Disassembly window, you can invoke the shortcut menu which allows you to easily set a breakpoint or watchpoint.

For software breakpoints, simply click on left of any source line or low-level address. Click on Breakpoints window in the View menu or Breakpoint dialog box in the Debug menu to display a list of tasks you may perform. Hitool also supports conditional breakpoint, allowing you to specify a condition under which the target will break.

Providing that the watchpoint is in use, it can be set to break on any address read, write, or access; any data read, write, or access; any instruction execution (RAM or ROM); or after any specified number of clock cycles.

Integrate "UTILS"

With Hitool, real-time target control is always reliable. Rather than having to jump from program to program, typing in commands, and locating files, everything you need to perform your debugging tasks is available from within Hitool.

Project-Wide 'Grep'

With its improved Windows 95 format, the Find / Find Next For Window allows you to perform project-wide searches or module-specific searches for text, cross-referencing variables easily.

Product Warranty

We provide a thirty-day limited warranty for hardware products. Defective equipment will be repaired or replaced as determined by Technical Support.

Software components are sold "as is" and defective repair is made on a best effort basis in priority order as determined by Technical Support.

An Extended Warranty is available for purchase and offers the added advantage of a firmware update service. Ask your local sales representative for details.

Related Publications

- See the Microsoft Windows 95, 98 or NT *User's Guide for the Microsoft Windows Operating System* for more information on using Windows, Notepad, and Macro Recorder.
- See the pertinent Processor User's Manual for information on the target processor you are using.

Registering

Be sure to complete and return the enclosed registration card so that Hitool can provide you with technical support and keep you informed of new product developments.

2 Getting Started

System Requirements

Currently, the host platforms available for Hitool are Windows 32-bit operating systems. Before installation, make sure that you have the following minimum host configuration:

Windows Requirements

- A personal computer with a Pentium or higher processor, with a CD-ROM drive to install Hitool for ARM software, documentation, and examples.
- Microsoft Windows 95, Windows 98, or Windows NT 4.0 operating system or later.
- A minimum of 32 MB of RAM, available disk space.
- VGA or Super VGA color monitor and Mouse.
- Printer or Serial port
- USB port (optional)

Installing Hitool for ARM

Hitool for Windows 9x/NT

Follow these steps to install Hitool for Windows 9x / Windows NT release on your hard drive.

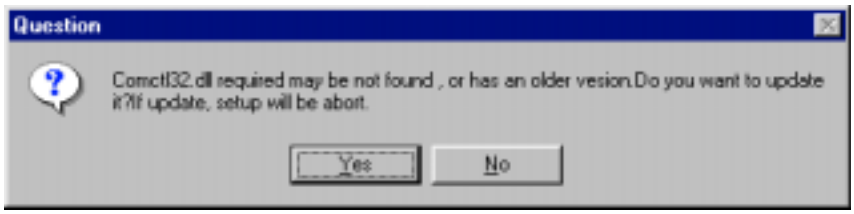
NOTE

Hitool runs under 32-bit Windows programs only - Windows 95 / 98 and Windows NT. Hitool does not support Windows 3.1.

Installing on Microsoft Windows 95/98 and Windows NT

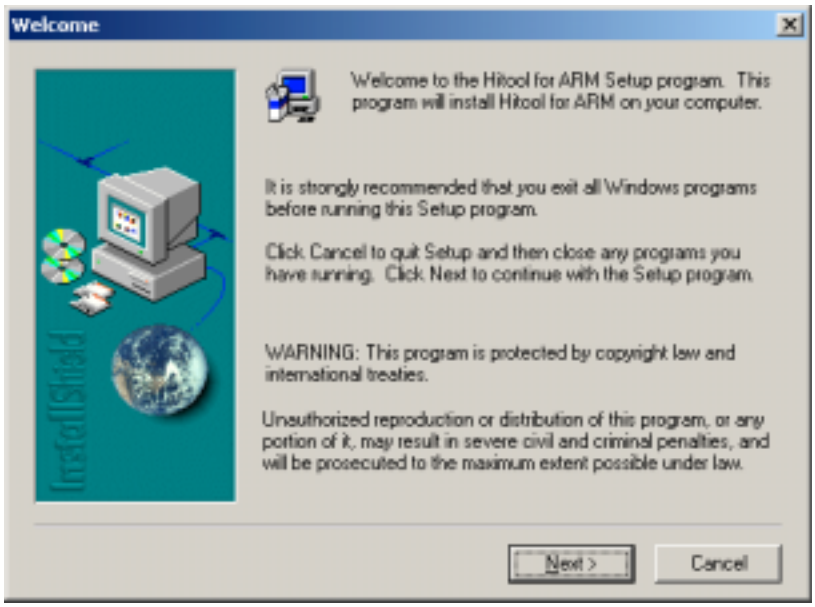
- 1. Invoke Microsoft Windows 95 / 98 / NT on your PC.
- 2. Place the Hitool disk in the proper drive.
- 3. Run Setup.exe.

If the comctl32.dll in your system is older than the one required, the following dialog box will popup.

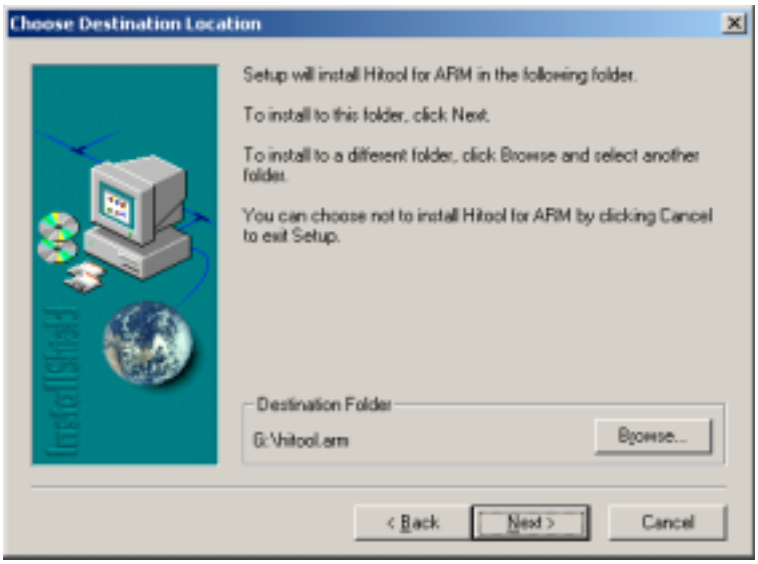

Question
Comctl32.dll required may be not found , or has an older vesion.Do you want to update it?If update, setup will be abort.
Yes
No

Setup will auto-run 401COMUPD.exe. Be sure to click on Yes button to update it.

After you restart computer, run Setup.exe to continue.


Welcome
Welcome to the Hitool for ARM Setup program. This program will install Hitool for ARM on your computer.
It is strongly recommended that you exit all Windows programs before running this Setup program.
Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.
WARNING: This program is protected by copyright law and international treaties.
Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.
InstallShield
Next >
Cancel

- 4. Click on Next button to go on.

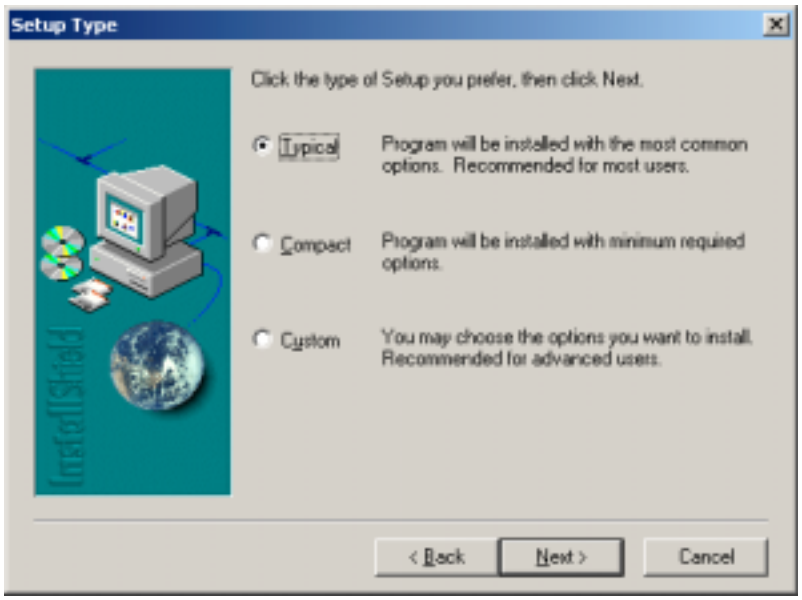

Choose Destination Location
Setup will install Hitool for ARM in the following folder.
To install to this folder, click Next.
To install to a different folder, click Browse and select another folder.
You can choose not to install Hitool for ARM by clicking Cancel to exit Setup.
Destination Folder
G:\Hitool.arm
Browse...
< Back
Next >
Cancel
InstallShield

- 5. After the setup program is loaded, you will need to indicate the directory where you want the release to be installed. By default, it will be installed in the following directory:

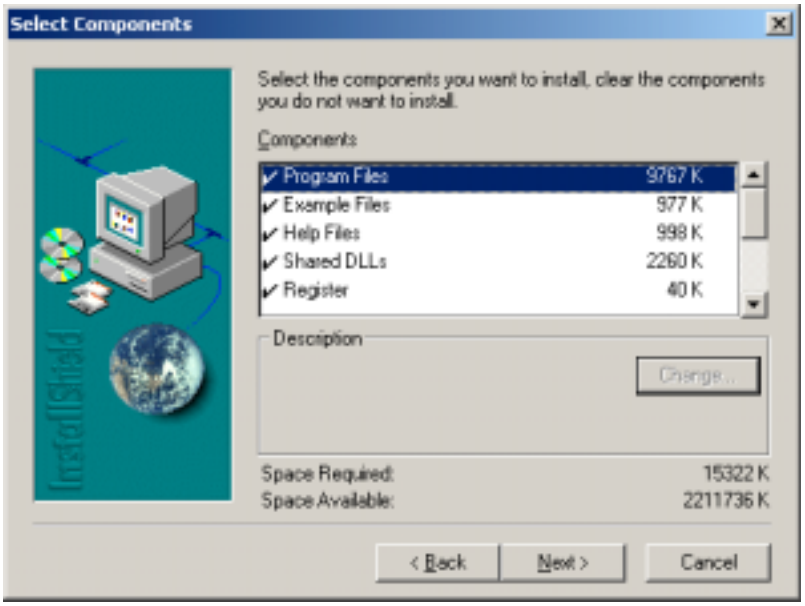
  C:\Hitool.ARM (where, C:\ is the system driver)

  Click on the OK button if the above default directory is the location where you want to install the release. We recommend that you use the default **C:\ Hitool.ARM** directory so that the changes you will need to make to the demo templates are minimal. Otherwise, you can edit the path as necessary. Then click on the OK button.

- 6. Select the type of Setup you would like to install, then click on Next button to go on.


Setup Type
Click the type of Setup you prefer, then click Next.
Typical
Program will be installed with the most common options. Recommended for most users.
Compact
Program will be installed with minimum required options.
Custom
You may choose the options you want to install. Recommended for advanced users.
InstallShield
< Back
Next >
Cancel

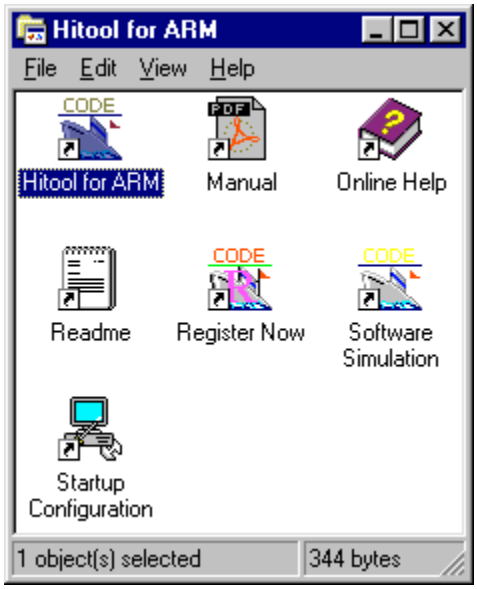
If you choose Typical type, most common components will be selected. That is default. If you choose Custom type, the Select Components dialog box will popup and you can select components in it.



In the remaining installation dialog boxes, click on Next button to choose default components.

The setup program will copy and decompress the files to the installation directory.

After installation is completed, Hitool for ARM Program Group will appear. Within this folder, you will find the Hitool application.



Hitool for ARM Program Group

About NT driver

MAC_MOT.sys is used as the parallel driver under NT or win2K.

The installation disks will install and start the driver automatically. But if user log on as non-administrator, the driver will not be installed for no privilege.

If Hitool for ARM reports the message of no driver, do the following:

1. Copy mac_mot.sys into <NT>\system32\drivers.

2. Run "dvrload mac_mot install start" in command line.

About Dongle

Hitool provides an USB dongle. You can launch Hitool for ARM on the PC with the dongle which has registered. For more details about registration, please refer to the following Registering section.

Further more, You can launch Hitool for ARM on other PC if it is with the dongle. In this case, you can register with the same serial number on the PC but need not to send a mail to Hitool. Simply copy the license file to the PC will be OK. For more detailed information about registering, please refer to the following Registering section.

To use the dongle, you should install the dongle driver first. The installation process will install the dongle automatically. Be sure to install Hitool for ARM software first, then insert the USB dongle into the USB interface. If you have trouble with the dongle, please restart PC to make sure that the dongle will work properly.

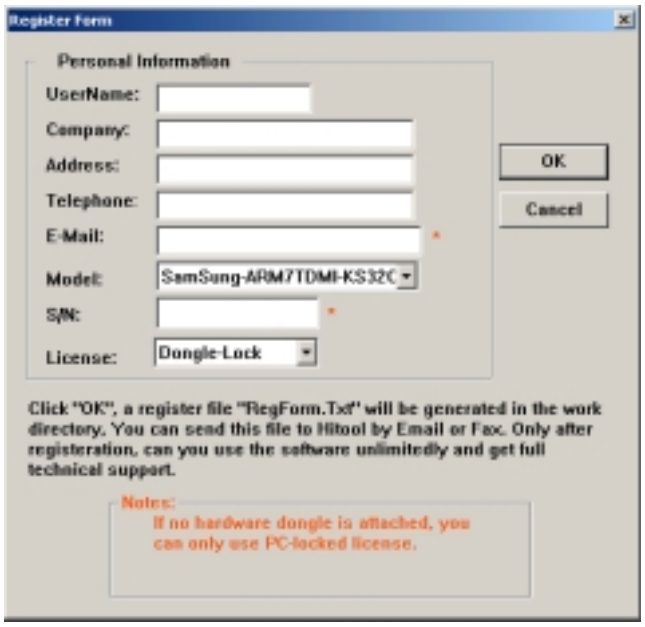
Registering

You can use Hitool for thirty days before registering. If you aren't a registered user, every time you launch Hitool a Register Information dialog box will popup as following:

Register Information
Hitool
This software has not been registered. You may evaluate this software for 30 days before registering.
If you want to apply a license, please press 'Register' button;else press 'Evaluate' to continue. After registering , you can use this software unlimitedly and get full technical support.
Register
Evaluate

If you haven't got the license, do the following to get it from Hitool:

- Click on Register button in the Register Information dialog box. In the popup Register Form dialog box fill in personal information then click on OK button. And a file named Regform.txt will generate in the current directory. Please send this file to Hitool by Email or fax.



If you have installed Windows MAPI, it will invoke default mail program which enable you to send mail to Hitool for the license. Be sure that your personal information is right. Hitool will respond your mail with the exact license file.

After you get the license file, you should save it to the installation directory, and then you can use Hitool without limitation.

Hitool accepts two type of license: Node-Lock and PC-Lock. If you select Node-Lock, you need an USB dongle. Please refer to the above section About Dongle for details. If you select PC-Lock, the license is only available for the PC who has registered.

NOTE

If you are a NT user, be sure to login NT as administrator. Otherwise, you can not register successfully.

Checking Host Configuration

Configuring Communications

Communication Parameters

Before you configure communications, it is important to be aware of the parameters to stay within while working with Hitool.

These parameters are as the following:

1. Parallel (LPT1 – LPT2) or Serial communications up to 115200 baud (COM1 -COM4).
2. SamSung, ATMEL or other maker and the corresponding Target CPU.
3. Wiggler, Raven, PowerProbe, PowerProbe+, Angel or Jeeni protocol or Simulator.

WARNING

1. It is meaningless that JEENI communicate with more than one debugger at a time. If two debuggers are simultaneously connected with JEENI, they will interfere with each other, causing unpredictable errors.

2. For Angel Protocol, Angel debug monitor on the board will occupy some memory region.

4. Little Endian or Big Endian according to the board you use.
5. Use initialize file to initialize. Hitool provides initsamp.inc sample file in ARM installation directory for your reference.
6. HeartBeat option.

NOTE

HeartBeat option is only available for Angel, Jeeni Protocol.

A debug initialization file contains a set of commands that initialize the target board when it is launched by the debugger.

A debug initialization file is a command file processed during Debug launch and each time **Reset** is selected in the **Debug** menu. A debug initialization file can perform several functions:

- Initialize registers and memory in targets that do not yet have initialization code in ROM.
- Other special initialization code.

To create a debug initialization file that contains a register section:

1).Use your data sheet to determine what your chip-select and pin-assignment registers should be following initialization.

2).Edit a copy of the appropriate “init.inc” file as plain (ASCII) text with any text editor or word processor in the include file syntax provided by the Hitool.

3).Define each register on a separate line to apply the correct values for your target.

6. Semihosting options, Vector address, Top RAM are only available for ARM toolchain.

NOTE

If the options about semihosting aren’t configured correctly, you can’t debug the code correctly.

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism could be used, for example, to allow functions in the C library, such as printf() and scanf(), to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

Semihosting is implemented by a set of defined software interrupt *(SWI*) operations. The application invokes the appropriate SWI and the debug agent then handles the SWI exception. The debug agent provides the required communication with the host. In many cases, the semihosting SWI will be invoked by code within library functions. The application can also invoke the semihosting SWI directly. Refer to the C library descriptions in the compiler manual for more information on support for semihosting in the ARM C library.

At this point, you can initialize communications.

Initializing Communications

To initialize communications, do the following in the Startup Configuration dialog box:

1. Select the Maker.
2. Select the Target CPU.
3. Select the Protocol.

NOTE

1. Choosing Simulator means running Hitool in simulator mode.

2. Wiggler/Raven/PowerProbe will not support ARM9 for hardware limitation.

4. Select your communication port supported by the selected protocol.
5. Set the port parameter corresponding to the selected port: Baudrate 9600, 19200, 38400, 57600, 115200 for COM port or address of parallel port.

NOTE

1. If you are using Hitool with Wiggler, Raven, PowerProbe or PowerProbe+ Protocol, you must be communicating over LPT1 / LPT2. For Angel or Jeeni Protocol, only communicating over Serial port (COM1 -COM4).

2. If LPT1 is used, make sure that its address is 0x378; if LPT2 is used, make sure that its address is 0x278. And be sure that the PC CMOS setting is consistent with the address of LPT. If the wiggler or raven work abnormally, try to change LPT mode. (ECP, EPP etc.)

3. When using parallel port, if "no driver" massage is displayed, simply run driver.bat in Hitool for ARM installation directory, which will install the mac_mot.sys and fdparse.sys automatically for your debugging.

6. Select the Endian.
7. Check to whether to Use initialize file.

For Angel Protocol, initialize file is grey because ANGEL agent has initialized the target when powering-on. Therefore writing some system perip register (ROMCON etc.) could cause ANGEL unpredictable.

And you'd better not to use 115200 baudrate. Under this baudrate, the communication may be unstable. ARM Debugger can not connect with the hardware under this baudrate at all. Some memory writing may be timeout, such as 0x8. It seems that ANGEL does not protect

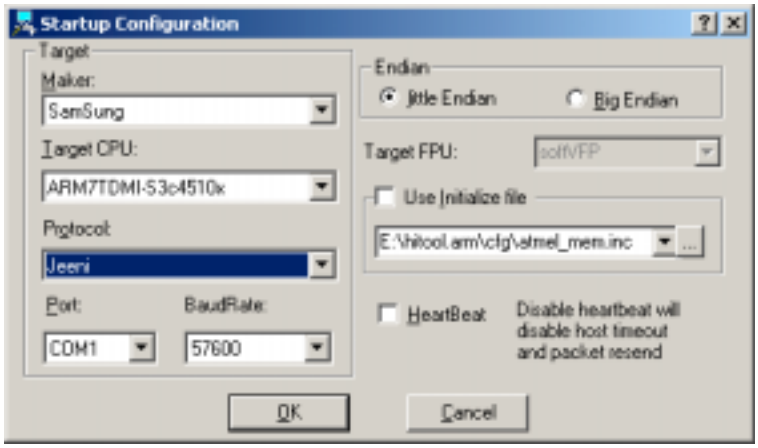
these region, but if these region are changed, ANGEL would be unpredictable.

8. Config the semihosting if necessary.

NOTE

For Wiggler, Raven, PowerProbe, PowerProbe+ Protocol and Simulator, set the semihosting in Startup Configuration dialog box.

For Angel, Jeeni Protocol, set the semihosting in Options dialog box Target Control tab. For detailed information, please refer to "Semihosting".



Now that you have selected your settings, you can launch Hitool for ARM, which would connect t target according the communication settings.

Once you have established communications, you need not to make configuration again when launch Hitool next time unless the hardware is changed.

Debugging your own project

In order to debug your own project or modify a saved project, you must install the Hitool release software.

To install the Hitool release software, follow the steps listed in the "Installing Hitool for ARM" section at the beginning of this chapter.

Once you have installed the software, you must configure communications. Follow the steps listed in the "Configuring Communications" section which appears earlier in this chapter.

Once you have completed these two steps, there are several additional steps you must complete before you can begin debugging. The following sections discuss the download formats.

Before you can download file, you will need to know the compatible download formats.

Downloadable Formats

Hitool debuggers accept the following download formats:

1. Motorola S Records (.s).
2. Elf Files (.elf, .axf, .out).

.s, .elf, .axf and .out are the load extension specified by the project settings (**Output Name**).

One of the first steps in the debugging process is to load the executable file to be debugged into the memory. You can load the executable file created by yourself or provided by Hitool whose default path is "\Hitool.ARM\examples".

Hitool IDE

Hitool is an Integrated Development Environment (IDE) for embedded system developers. It provides strong features for debugging, editing and project management. In addition, Hitool also provides other utilities to facilitate your project development, such as include batch, macro etc.

Project Management

Hitool is based on Project Management. The Project Manager enables you to construct necessary environment to automate the process of debugging or building.

Hitool adopts the concept of project to maintain information about the system you are debugging or building. A project describes how to load an executable file, set breakpoint, how to build executable file etc.

Hitool debugging actions start from opening project. Opening a project means to restore all the environment you have set previously and auto-load the executable file into memory. To make it easy to use,

Hitool also provides Load function. When you load an executable file directly, Hitool will auto-open/new a project first.

Its project management further lies in project setting saving. When you close a project or open another project, all the elements such as breakpoints, watchpoints and so on will be saved automatically with the project. If you open the project next time, all the settings will be restored.

Also Hitool permits you to build your own executable file. You can refer to Building Project in Chapter three for more detailed information about building. In the project, a portion of project setting controls how to build it. By default, no source file is added. So if you want to build your own executable file, please add the source files manually.

IDE

The Hitool IDE (Integrated Development Environment) is the center of Hitool software development. The IDE offers:

Power - From the IDE you can edit, browse, compile, link, and run your project. The IDE includes a text editor and a project manager to help you do all your programming tasks easily and quickly.

The IDE's core tools are the tightly and transparently integrated primary components used to write code. The following table describes some of the individual components included with the Hitool IDE.

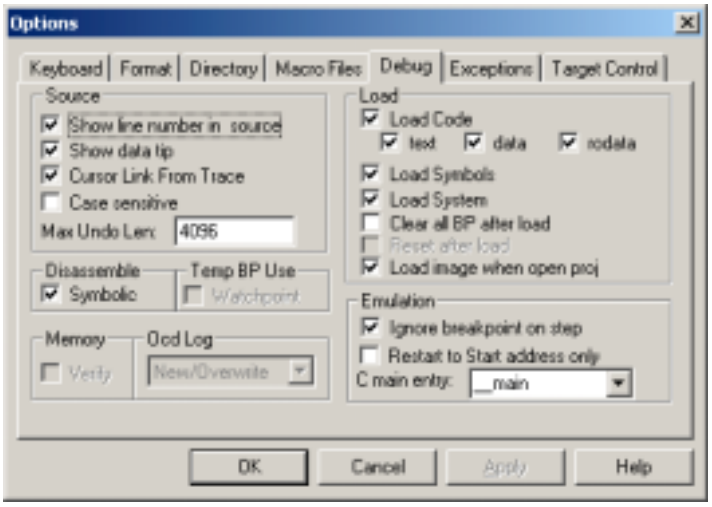
This tool…	Provides these features and functions
Project manager	Use the commands in the **Project** menu to gather all the files of a program together with configurable sets of options into a project file. These commands decide for you which plug-in compiler and linker to act on the project to create the final executable file.
Source code editor	Use the commands in the **Edit** menu and the controls in the IDE's editor windows to edit source code and text. Some of the editor's features are: • edit, search, and replace text in one file or several files • split an editor window into multiple panes

	• set bookmarks at arbitrary locations in a source file • compare two files or folders.
Compiler and linker	Thanks to its plug-in architecture, the IDE is able to play host to GNU compilers and linkers to generate software for several operating system and processor platforms.
Debugger	Use the commands in the **Debug** menu to control your program, step through its source code or assembly statements, set breakpoints, set watchpoints and examine its data while it executes. The IDE's debugger works with all kinds of code in all the supported languages--C, C++. Use the debugger to debug libraries, executables, multiple threads and processes. The debugger also supports debugging code running on multiple processors.

Load Options

Clicking on the Option menu item from the Option menu will invoke the Options dialog box.

The following Debug tab in this dialog box provides access to debug options, including load options.



Load Options

Below is a listing and brief explanation of the available options offered in the Debug tab Load section.

- **Load text / data / rodata** – After building, the code of .elf format file usually consists of three sections, text, data and rodata. Checks to download text, data and rodata section and text, data are compulsory.
- **Load Symbols** – Check on this item and each time your image is downloaded the symbols will be downloaded to provide debugging information.
- **Load System** – This item specifies that system variable symbols would be downloaded.
- **Clear all BP after load** – Check on this item and the system will delete all the breakpoins you have set afer loading the project again.
- **Load Image when open proj** – This item specify that when you open a project the load information about the project will be showed.

Load Type

Hitool provides you two types for load image and symbols. Click on Load Image or Load Symbol Only menu item from the File menu, then in the pop up the Load File dialog box choose the executable file.

Below is a listing and brief explanation of the two types offered in the File menu.

- **Load Image** – Standard load. Depending on the settings in the option dialog box.
- **Load Symbol Only** – Loading symbols only.
- **Load Code Only** – Keeping current symbols and loading new code only.

Downloading the Binary Image and Reading Symbols

Once you have established communications, you could download your symbol and code.

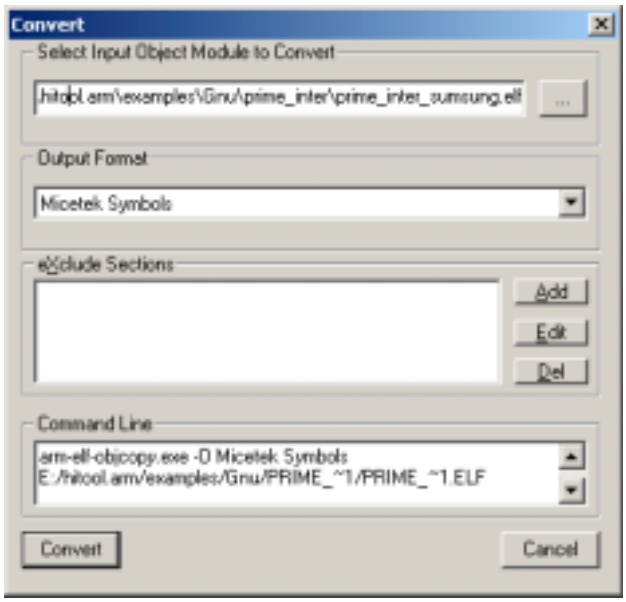
NOTE

Before loading code, Hitool will check whether the Endian of elf file is the same as the Endian of hardware you use. If not, we will give error message box. If you debug in simulator mode, you may build the executable file in Little Endian format. If you only load symbols, Big Endian or Little Endian are both OK.

After you loading the executable file successfully with symbol information for the first time, you will find two new files, .ht file and .st file generated in the load path whose name are the same as the executable file name. The two files will be used while you reload the executable file to improve the speed of loading symbol especially for the large size files. Every time you load an executable file, Hitool will try to find out the two files and compare them with the executable file. If Hitool can not find both two files or the executable file is newer, Hitool will use the executable file itself to load symbol information and re-generate .st and .ht files. Otherwise Hitool will use the two files instead of the executable file to load useful symbol information.

You can generate the two files manually. To do this,

1. Choose Convert from Tools menu. Convert dialog box will pop-up as the following:



2. Select the executable file to convert.
3. Set the Output Format as Micetek Symbols from the list box.
4. Click on Convert button.

.ht file and .st file will generate in the current path whose name are the same as the executable file name.

Hitool offers several methods of loading your executable file as described below.

Load from List of Recently Projects

Because of project management, an executable file is assigned to a project. So you can open the project by choosing it from the bottom of Project menu to load it automatically. Hitool lists up to four most recently used project files.

Load with Load Command or Load/Reload Button

ReLoad Button

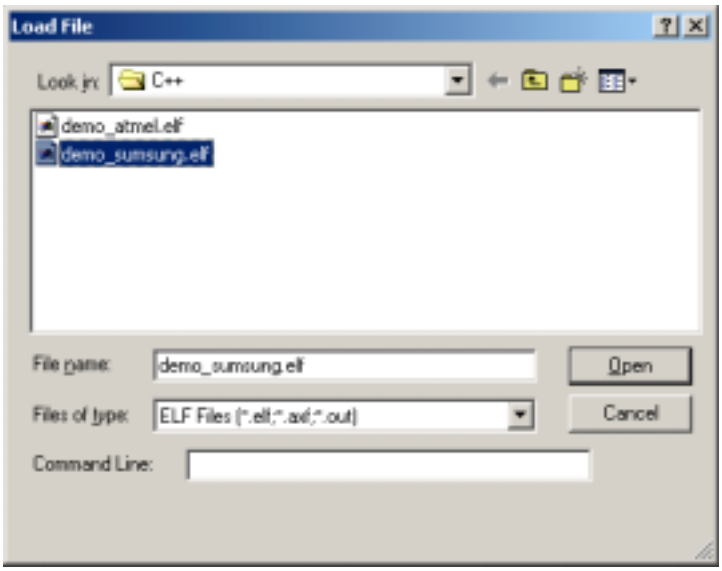
When you launch Hitool, Load File dialog box (as illustrated below) will popup automatically, you could choose an executable file in this dialog box and click on Open button.

You can download the code by the load command or load/reload button whenever.

If you haven't opened a project, click on Reload button is equivalent to click on load button.

Load Button

If you have opened a project, choose Reload command from the Project menu or click on Reload button on the debug Toolbar to reload it simply. To load another executable file, please click on the Load button on the debug Toolbar to load it with the corresponding project opened first.



Load File Dialog Box

Project Parameters

Download: In the Load File dialog box appears, enter your object file, or double-click to select an executable file from your system. Then click on the Open button to download the file.

Source Files: Enter your source paths in the Options dialog box Directory tab. Using the buttons, you may edit search paths, enter multiple paths to be searched.

When Hitool attempts to browse a source module in the Source Window, it follows the following search orders:

- The source path specified in the debug information contained in the executable file.
- The source specified in the project Source Files folder.
- The Source Files path specified in the Options dialog box directory tab.
- The path specified by the user directly.

Source Path information and related options are discussed in greater details in the chapter entitled “Viewing Source Code” found later in this manual.

Troubleshooting Tips

This section provides information about how to diagnose and fix the associated problems. The following types of problems are:

- Reading symbol information into Hitool
- What to do when you are unable to establish valid target communications
- Diagnosing download problems

As always, if you encounter any difficulties, try to find the answer in the user's manual or on-line Help. If you can't find the answer, contact Hitool Technical Support Department.

3 Configuring a Project and Building Target

Building Target

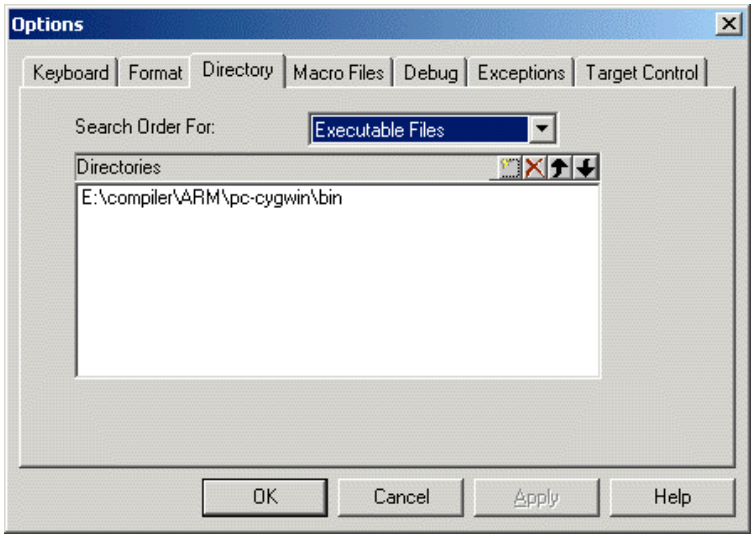
This section describes how to build target.

Configuring Compiler Path

Now let's begin with a sample project now. You can use GNU compiler to generate an executable file for debugging.

> ***NOTE***
>
> *GNU compiler isn't installed with Hitool for ARM installation. So before using GNU compiler, you should install it in addition. The default installation directory is <Hitool for ARM installation directory>\compiler.*

Be sure to configure compiler path in the beginning. To do this, from Option menu choose Option command. Switch to Directory tab in the Options dialog box that popup, then set the path of executable files, global include file path and lib file path as the following:



NOTE

1. *Be sure to re-specify the path of the Executable Files when you change the compiler installation directory. And the paths of global include files path and lib files path will also be changed manually.*
2. *If you install the Gcc compiler with the installation disk we provided, please make sure that make.exe and ARM-ELF-GCC.exe are included in the Executable Files directory you sepecified.*

Creating a new project

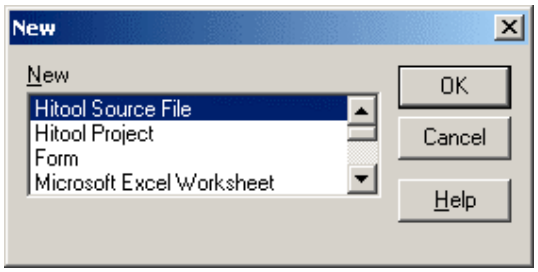
Hitool provides Wizard dialog box to guide you create a new project step by step which we call AppWizard.

New a Project

To new a project, you can choose one of the following and the following New Project dialog box will pop up:

1. Click on New menu item from File menu.
2. Click on New button on standard toolbar.

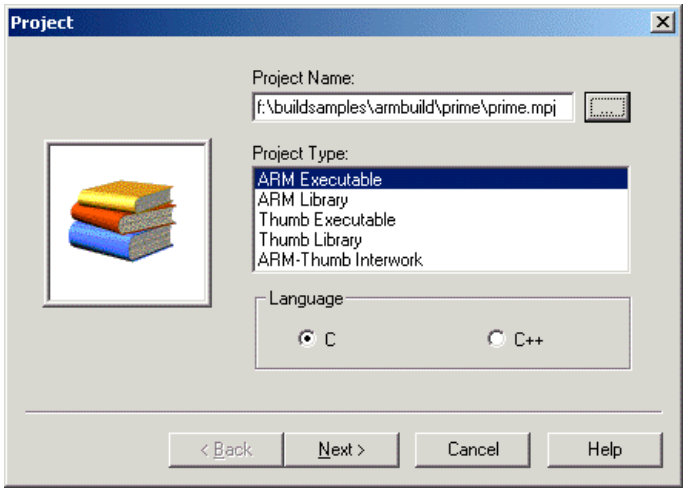
Then select Hitool Project from the following pop-up New dialog box:



-or-

3. Click on New command from Project menu.

Project page will pop-up as the following:



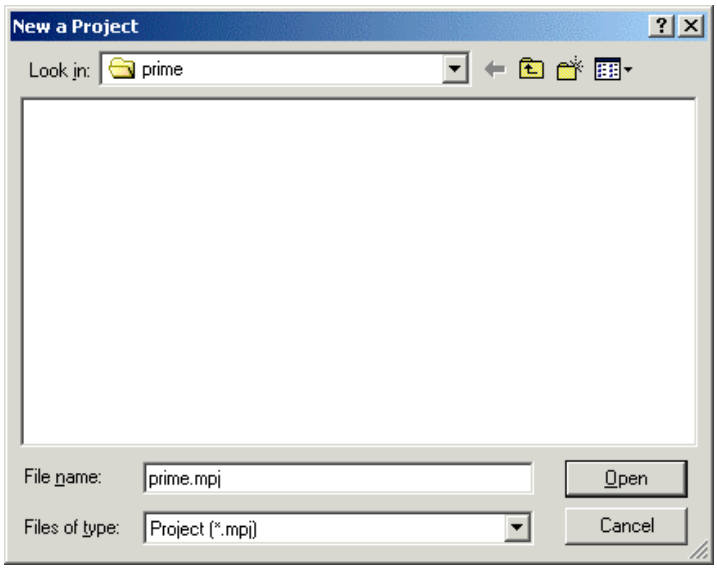
Project Page

Specify Project Path and Name

You should specify the name and path of a blank project in the popup Project page first.

There are two controls for this option: an edit box and a button.

You can enter the project file path and name in the edit box directly or click on the "..." button. If you click on the "..." button, New Project dialog box will pop-up as the following in which you can set the project file path and fill in a name serve as your project file name. The default extension of the project file is ".mpj".



New Project dialog box

Set Project Type

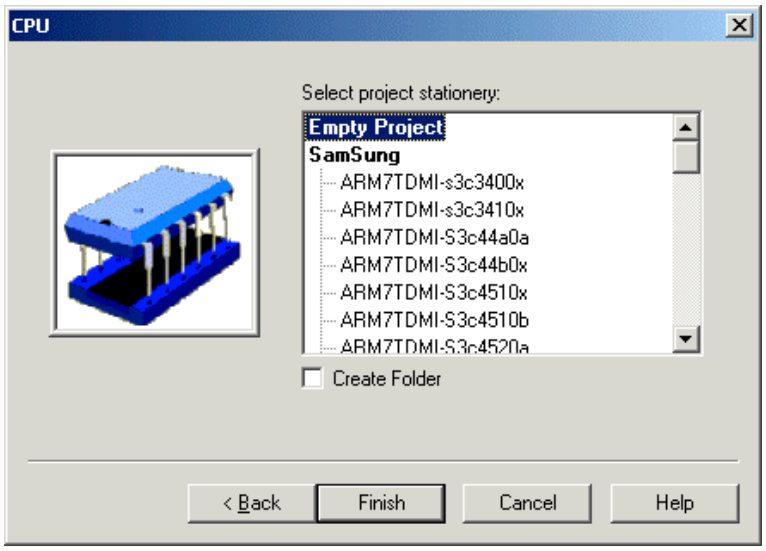
Choose Project Type then Language. Hitool provides five types of Project Type, ARM Executable, ARM Library; Thumb Executable, Thumb Library and ARM-Thumb Interwork; two types of Language, C and C++.

Click on Next button in the dialog box to continue.

Select Project Stationery

In the following pop-up CPU page, you will see a tree listing supported CPU. If you select "Empty Project", AppWizard will just generate a single project file (.mpj). Otherwise, If you select a leaf node of the tree, AppWizard will automatically generate two files in the current directory: main.c and start_up.s. After finish the setting, the three files will be added into the project automatically.

Furthermore, there is a option Create Folder for your check. If you check it, the project will not be generated simply in the path you specified, there will be a new folder named after your project to store the project and relative files.



CPU Page

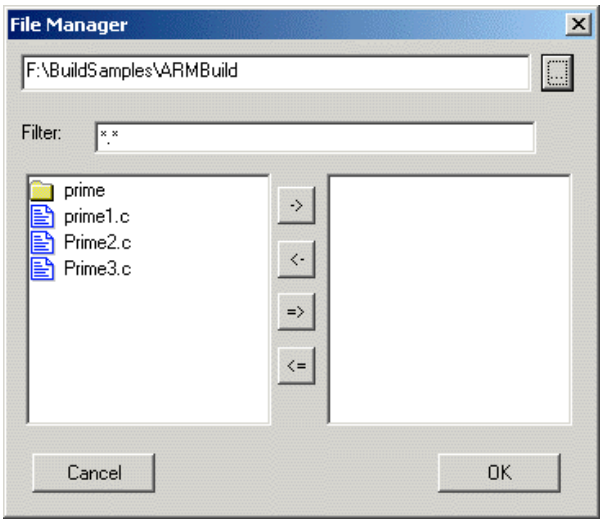
Click on Finish button.

Choose files to add

After the AppWizard dialog close, a file manager dialog pops up. The left list box lists the file names. You can select one or more source files or header file to add into the project or remove the files. You can select none to configure the project later.

NOTE

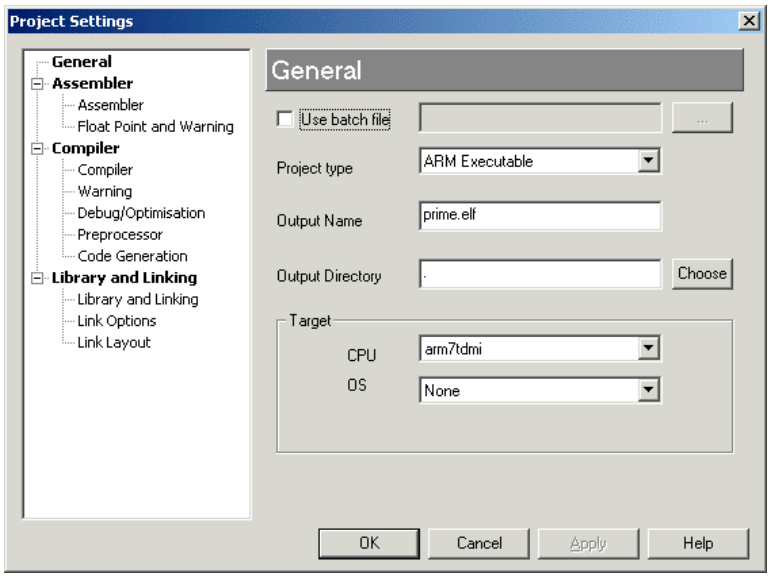
Make sure that the source files and the project must be in the same path.



Click on OK button in the dialog box.

Configuring Attributes of the Project

After you specify the project name, a project name is created automatically. At the same time, Project Settings dialog box will pop-up automatically as the following:



Project Settings dialog box

The Project Settings dialog box handles settings that affect how the Hitool IDE builds a specific target file within a project.

Simply click on OK button in Project Settings dialog box to close it. Hitool will create .ld file and .mak file automatically in the current directory whose names are both the same as the project file name.

Adding Source Files

You can add your source files into the project via one of the following ways:

1. Click on Add Source Files menu item from Project menu.
2. Click on Add Source button on the Build toolbar.
3. Right click on the folder in Files pane in the Project window and choose Add Source Files... item from the pop-up menu.

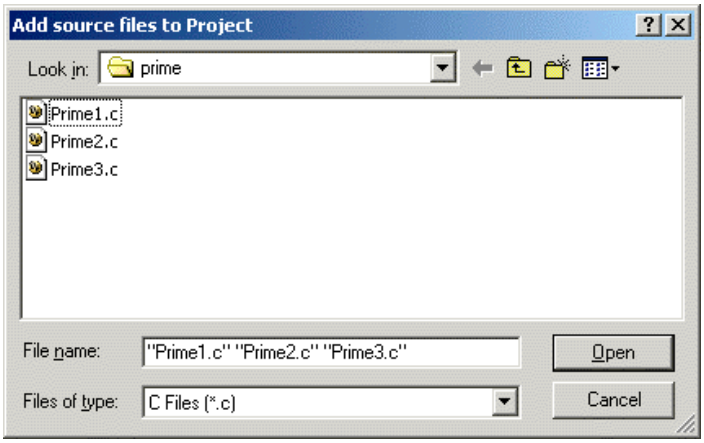
NOTE

If the project type you choose is ARM-Thumb Interwork, you can only use the third way mentioned above to add source files.

Add source files to Project dialog box will then pop-up as the following. You can choose one or more files at one time and add them into the project.

NOTE

Be sure not to choose the source files in the long path directory with parentheses.



Add source files to Project dialog box

In order to initialize the program correctly, you are also required to add a start-up file into the project besides the source files.

Adding START_UP File to Project

In the examples Hitool provides you can see there is a file named "demo_start.s" in every project. It is the right start-up file. In addition, there is a header file "arm.h" at the same directory which "demo_start.s" needs during compiling.

The following is the detailed list of example "arm.h".

```
#ifndef          __ARM_H_
#define          __ARM_H_

#define IRQ  0
#define FIQ  1

#define Mode_USR                0x10
#define Mode_FIQ                0x11
#define Mode_IRQ                0x12
#define Mode_SVC                0x13
#define Mode_ABT                0x17
#define Mode_UNDEF              0x1B
#define Mode_SYS                0x1F
#define I_Bit                   0x80
#define F_Bit                   0x40
```

```
#define SVC_Stack                    0x10000
#define ABT_Stack                    0x1000
#define UNDEF_Stack                  0x1000
#define IRQ_Stack                    0x5000
#define FIQ_Stack                    0x5000
#define USR_Stack                    0x1000
#define SYS_Stack                    0x10000
#endif
```

Next we will give you the detailed list of example "demo_start.s" as the following:

```
#include "arm.h"

        .text
.global _start
_start:
 BL  Init_Stack
 B  __main

Init_Stack:

MOV  R7, LR
 LDR  R1, =_stackBottom

MOV  R0, #(Mode_IRQ | I_Bit | F_Bit) @ No interrupts
 MSR  cpsr, R0
 MOV  SP, R1
 SUB  R1, R1, #IRQ_Stack

MOV  R0, #(Mode_FIQ | I_Bit | F_Bit) @ No interrupts
 MSR  cpsr, R0
 MOV  SP, R1
 SUB  R1, R1, #FIQ_Stack

MOV  R0, #(Mode_ABT | I_Bit | F_Bit) @ No interrupts
 MSR  cpsr, R0
```

```
 MOV  SP, R1
 SUB  R1, R1, #ABT_Stack

MOV  R0, #(Mode_UNDEF | I_Bit | F_Bit) @ No interrupts
 MSR  cpsr, R0
 MOV  SP, R1
 SUB  R1, R1, #UNDEF_Stack

MOV  R0, #(Mode_SYS | I_Bit | F_Bit) @ No interrupts
 MSR  cpsr, R0
 MOV  SP, R1
 SUB  R1, R1, #SYS_Stack

MOV  R0, #(Mode_SVC | I_Bit | F_Bit) @ No interrupts
 MSR  cpsr, R0
 MOV  SP, R1
 SUB  R1, R1, #SVC_Stack
 MOV  R4, R1

MOV  R2, #0x0
 LDR  R0, =_fbss
 LDR  R1, =end

Do_Clear:
 STR  R2, [R0], #0x4
 CMP  R0, R1
 BCC  Do_Clear

MOV  PC, R7
```

Demo_star.s can implement the following functions:

1. Initialize the stacks used by program.

2. Go to the main function in user's source file.

During edit demo_star.s, you should pay more attention to the following important factors:

1. "_start" label

In the beginning of demo_start.s, there is a global label "_start". It is the entry point of the whole program. You can apply the entry point with another name, such as Begin, Go, ToDo and so on.

NOTE

The name you applied must be consistent with the Image entry point option in the Link Options panel in the Project Settings dialog box.

2. "__main" label

After initialized the stacks, it will perform "BL __main" line. "__main" here is a symbol which marks the entry point of user's program. That is, if your source starts with "Begin" function as below:

```
int Begin(int i)
{
   if (i==0) {
         printf("%d", i);
         return 1;
   }
   else if(i<0)
      Begin(i++);
   else if(i>0)
      Begin(i--);

}
```

You must replace "__main" with "__Begin" which keeps the function name consistent.

3. Stacks Sizes

In arm.h header file you can see there are seven stacks and their default stack sizes:

SVC_Stack	0x10000 bytes
ABT_Stack	0x1000 bytes
UNDEF_Stack	0x1000 bytes
IRQ_Stack	0x5000 bytes
FIQ_Stack	0x5000 bytes

USR_Stack	0x1000 bytes
SYS_Stack	0x10000 bytes

The default sizes of these stacks are recommended, not compulsory. You can resize them according to personal requirement.

> ***NOTE***
>
> *The total amount of these stacks sizes should not exceed the "end" label (The end of program sections).*

That is, we allocate the stacks from "_stackBottom" (Higher address) to "end" (Lower address). If the space between "end" and"_stackBottom" is not large enough to hold all the stacks, an underlying error will be generated. The error resulted in during debugging cannot be detected during building. So be sure to estimate and specify the stacks' size carefully before building.

4. "_stackBottom"

_stackBottom is a symbol that tells the base address of stacks. You can specify it in Stack Top option in Link Listing panel in Project Settings dialog box.

In order to provide stacks with plenty of space, we strongly recommend you to specify it to the highest address of RAM. And this value must be word-aligned. Otherwise, there will be running error only during debugging.

After finished adding source files, you can begin building project.

Building Project

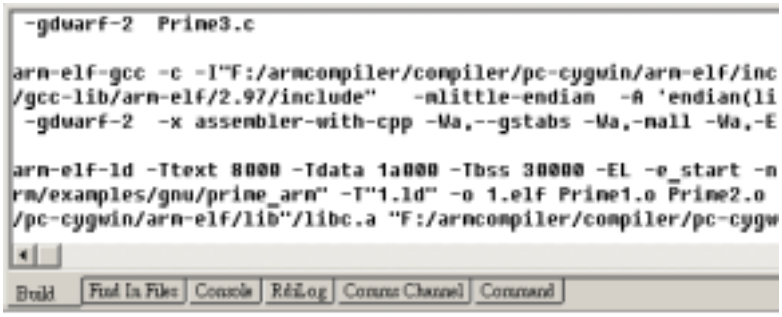
Do one of the following to build project:

1. Click on Build button on Build toolbar.

2. Click on Build item on Project menu.

3. Click on Rebuild All item on Project menu to rebuild the project.

The result is displayed in the following Build pane.



Output window Build pane

To stop building, simply click on Build button on Build Toolbar.

The lines shown in red will provide detailed information if errors exist. To locate to the corresponding source quickly, simply double click on the line. You can debug the source file directly without exit Hitool.

NOTE

If you build an opened project, please open the Project Settings dialog box to ensure the options correct especially to the Output Directory.

The difference between build and rebuild all is that when you build a project, compiler will compare the source files with the object file. Only when the source files are more up-to-the-minute will the compiler build to generate a latest object file. While during rebuild all compiler deletes the existing object file first and then generates it again, making no comparison.

If building succeeds, a target file will be generated in the output directory specified in the Project Settings dialog. The whole process of creating a project succeeded.

You can begin debugging Now.

Overview of the Project Settings Dialog Box

The following section describes how to configure build target options, including Compiler, Assembler and Linker options, for a specific build target in a project. Build target options specify how the Hitool IDE should process a build target in a project.

- Using the Project Settings Dialog Box
- Configuring General Target Options
- Configuring Assembler Settings
- Configuring Compiler Settings
- Configuring Library and Linker Settings
- Using the Equivalent Command Line text box

NOTE

1. *The settings you define in these panels apply to the currently selected build target only. You must configure each build target in your project separately.*
2. *Hitool doesn't provide GNU compiler. For use of compiler, you should install GNU compiler in addition.*

Using the Project Settings dialog box

This section gives basic information on using the Project Settings dialog box to configure Target options. When you change target settings, the changes you make apply to the currently selected build target in the current project.

Displaying Project Settings dialog box panels

Do the following to display Project Settings dialog box for a specific build target in the current project:

1. Open the project file you want to configure.

then

1. Click on Settings...item in the Project menu

-or-

2. Click on Settings button on Build toolbar.

Hitool IDE displays Project Settings dialog box with a list of available panels on the left. The options of selected panel in the list are displayed on the right in details.

3. Select the panel in the list you want to configure. To do this, you can use the up and down arrow keys or simply click on the name of the panel.

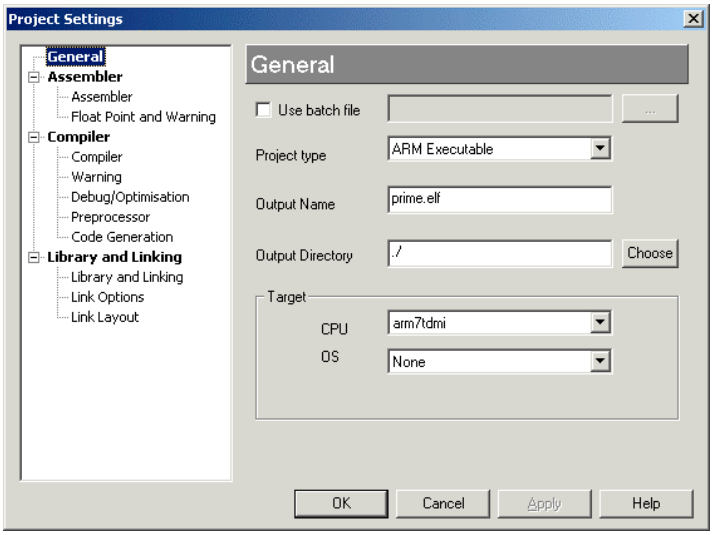
Each panel contains the related options you can configure. The options you set will apply to the current selected build target in the project.

4. Select the options as you require. See the following sections in this chapter for detailed descriptions of the options in each configuration panel.
5. Save or discard your changes, as required.

There are four buttons OK, Cancel, Apply and Help on the dialog box. You can save change by clicking on OK button, or discard change by clicking on Cancel button. When you make change in the dialog, Apply button becomes active. You can save change temporarily by clicking on Apply button. But if you click on Cancel button the temporary change you made are discarded. You can even get online help by clicking on Help button.

Configuring General Target Options

This section describes how to configure general options for a specific build target in the following General panel of Project Settings dialog box, such as the type of the project, the name of the output file and the directory of output file.



Project Type

There is a combo box to let you choose the type of the project. In the combo box there are five types for selection: ARM Executive, ARM

Library, Thumb Executive, Thumb Library and ARM-Thumb Interwork. You can select one of them to serve as the type of your project.

Output Name

In this edit box you may enter the desired output name. The default is a file whose name is the same as the project's name and the file extension is .elf.

Output Directory

This is an edit box. You can enter the directory either by keying in directly or clicking on the button besides it to select an existed path from the pop-up Browse for Folder dialog box.

> ***NOTE***
>
> *You must enter an existed directory, otherwise an error message box will pop-up advising you to re-select an existed path.*

Target

CPU	Choose the target CPU.
OS	Choose the target OS.

Use Batch File

There are three controls for this option: a check box, an edit box and a button.

If you set the check box, the edit box and the button are both enabled. You can enter the batch file's name in the edit box or select it by clicking on the "..." button. Also you could type the make file name .mak as batch file directly.

> ***NOTE***
>
> *If you check "Use batch file" option, you should edit the batch file by yourself.*

That means you don't want to use the options about the compiler in this Project Setting dialog box. So if you use "Use batch file" mode, you should prepare a batch file to control building process in advance.

Here we give you a sample batch file.

GNU compiler is installed in <Hitool for ARM installation directory> \compiler directory, so edit prime.bat as the following with any text edit tools:

SET PATH=<Hitool for ARM installation directory> \compiler\pc-cygwin\BIN;%PATH%

Set appropriate compiling options: Assembler, Compiler, Linker and Library, so continue to edit prime.bat as the following:

arm-elf-gcc -mlittle-endian -A -fno-rtti -DMACHINE_TYPE_ARM -w -O0 -g -gdwarf -I..\src_c\ -g ..\src_c\prime1.c -o prime1.o

arm-elf-gcc -c -Wa,-EL -gdwarf -DMACHINE_TYPE_ARM -I..\src_c\ -g ..\src_c\prime2.c -o prime2.o

arm-elf-gcc -c -Wa,-EL -gdwarf -DMACHINE_TYPE_ARM -I..\src_c\ -g ..\src_c\prime3.c -o prime3.o

arm-elf-gcc -c -fno-rtti -DMACHINE_TYPE_ARM -w -g -gdwarf -O0 -x assembler-with-cpp -Wa,--gstabs -Wa,-mall -Wa,-EL EL -I..\src_c\ ..\src_c\start_up.s

arm-elf-ld -Ttext 80000000 -Tdata 80100000 -e_start -EL -nostartupfiles -static -g -n -N –L. -Tprime.ld -o prime.elf start_up.o Prime2.o Prime3.o Prime1.o rs232.a ..../pc-cygwin/arm-elf/lib/libc.a ..../pc-cygwin/lib/gcc-lib/arm-elf/libgcc.a

Be sure that the batch file is in the same directory as the project file.

Configuring Assembler Settings

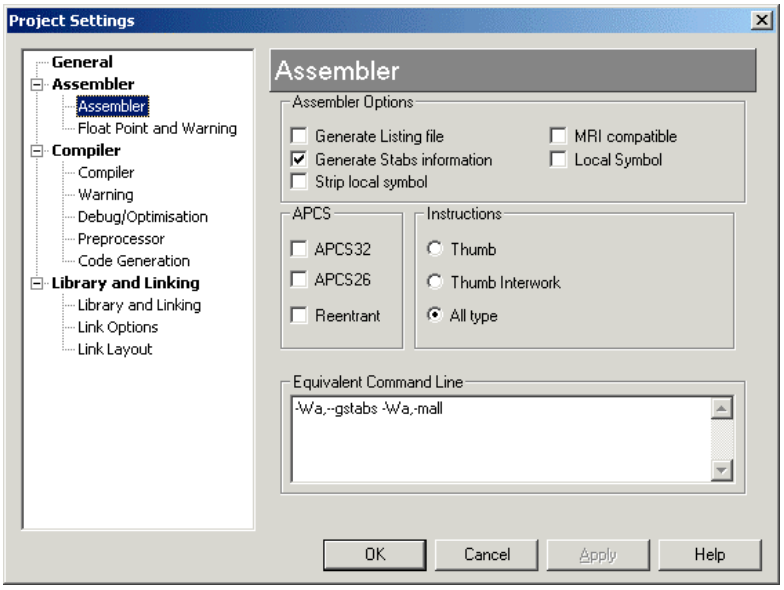
This section describes the assembler settings group of panels. These panels provide configuration options that are specific to the GNU assembler. Assembler use these panels to configure options for the assembler, including:

- Assembler
- Float Point options and Warning

Configuring the Assembler

Use the Project Settings to configure options for the Assembler:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Assembler in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Assembler panel of Project Settings dialog box as follows:



Assembler Options

Generate Listing file

Turn on listings.

MRI compatible

Assemble in MRI compatibility mode.

Generate Stabs information

Generate stabs debugging information.

Local Symbol

Keep local symbols.

Strip local symbol

Strip local absolute symbols.

APCS

APCS32

Specify 32 ARM Procedure Calling Standard is in use.

APCS26

Specify 26 ARM Procedure Calling Standard is in use.

Reentrant

Position independent/reentrant code has been generated.

Instructions

Thumb

Only allow Thumb instructions.

Thumb Interwork

Mark the assembled code as supporting interworking.

All Type

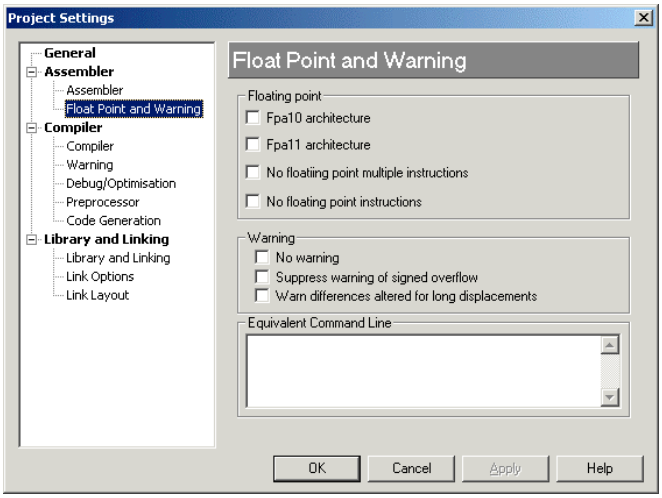
Allow any instruction.

4. Click on Apply button to save your changes temporarily.

Configuring the Float Point and Warning

Use the Project Settings to configure float point and warning options for the assembler:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Float Point and Warning in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Float Point and Warning panel of Project Settings dialog box as follows:



Floating Point

Fpa10 architecture

Select floating point architecture 10.

Fpa11 architecture

Select floating point architecture 11.

No floating point multiple instructions

Don't allow floating-point multiple instructions.

No floating point instructions

Don't allow any floating-point instructions.

Warning

No warning

Ignored.

Suppress warning of signed overflow

Don't warn about signed overflow.

Warn differences altered for long displacements

Warn when differences altered for long displacements.

4. Click on Apply button to save your changes temporarily.

Configuring Compiler Settings

This section describes the compiler settings group of panels. These panels provide configuration options that are specific to the GNU compiler:

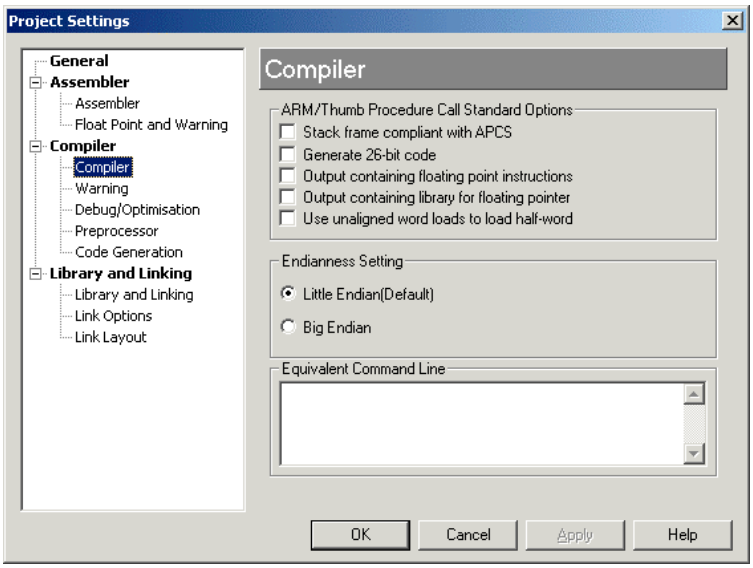
Use this panel to configure options for the GNU compiler, including:

- Compiler
- Warning
- Debug/Optimization Preprocessor
- Preprocessor
- Code Generation

Configuring the Compiler

Use the Project Settings to configure options for the compiler:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Compiler in the Project Settings dialog panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Compiler panel of Project Settings dialog box as follows:



ARM/Thumb Procedure Call Standard Options:

Stack frame compliant with APCS

Generate a stack frame that is compliant with the ARM Procedure Call Standard for all functions, even if this is not strictly necessary for correct execution of the code.

Generate 26-bit code

Generate code for a processor running with a 26-bit program counter, and conforming to the function calling standards for the APCS 26-bit option.

Output containing floating point instructions

Generate output containing floating point instructions.

Output containing library for floating pointer

Generate output containing library calls for floating point.

Use unaligned word loads to load half-word

Do not try to load half-words by loading a word from an unaligned address.

Endianness Setting

Little Endian (Default)

Little Endian.

Big Endian

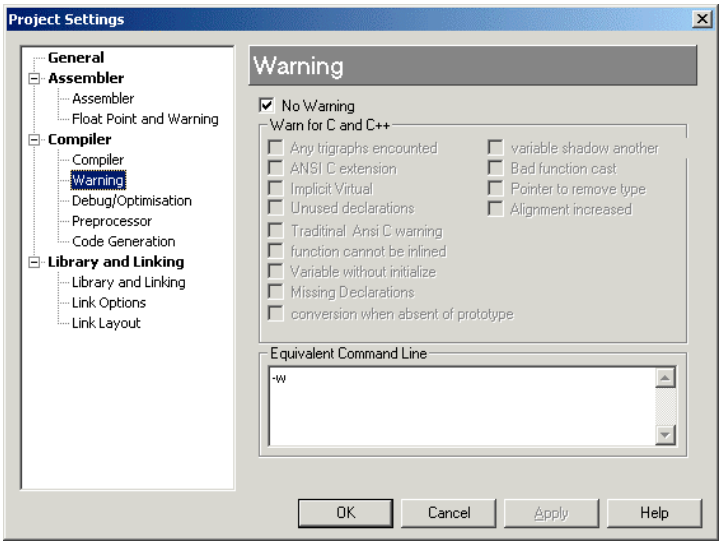
Big Endian.

4. Click on Apply button to save your changes temporarily.

Configuring the Warning

Use the Project Settings to configure options for warning:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Warning in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Warning panel of Project Settings dialog box as follows:



No Warnings

Directs the compiler to inhibit all warning messages.

Warn for C and C++

Any trigraphs encounted

Warn if any trigraphs are encountered.

Variable shadow another

Warn whenever a local variable shadows another local variable.

ANSI C extension

Issue all the warnings demanded by strict ANSI standard C, reject all programs that use forbidden extensions.

Bad function cast

Warn whenever a function call is cast to a non-matching type.

Implicit Virtual

Warn when a derived class function declaration may be an error in defining a virtual function.

Pointer to remove type

Warn whenever a pointer is cast so as to remove a type qualifier from the target type.

Unused declarations

Warn whenever a variable is unused aside from its declaration, whenever a function is declared static but never defined, whenever a label is declared but not used, and whenever a statement computes a result that is explicitly not used.

Alignment increased

Warn whenever a pointer is cast such that the required alignment of the target is increased.

Traditional ANSI C Warning

Warn about certain constructs that behave differently in traditional and ANSI C.

Function cannot be inlined

Warn if a function can not be inlined, and either it was declared as inline, or else the '-finline-functions' option was given.

Variable without initialize

An automatic variable is used without first being initialized.

Missing Declarations

Warn if a global function is defined without a previous declaration.

Conversion when absent of prototype

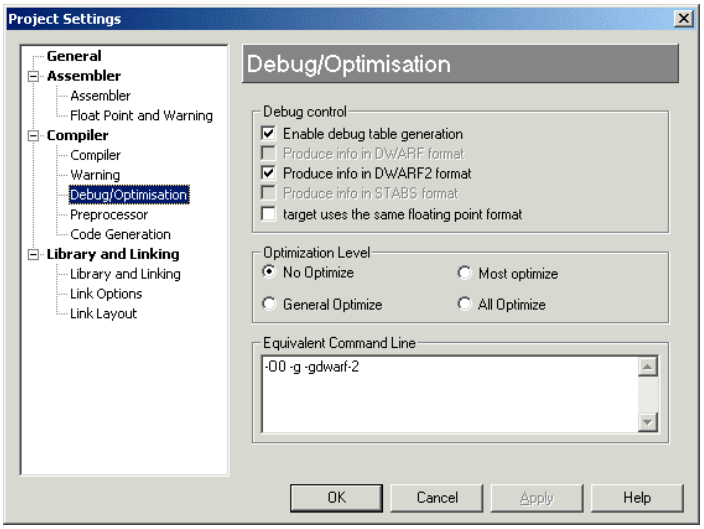
Warn if a prototype causes a type conversion that is different from what would happen to the same argument in the absence of a prototype.

4. Click on Apply button to save your changes temporarily.

Configuring the Debug and Optimization

Use the Project Settings to configure options for debug and optimization:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Debug/Optimization in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Debug and Optimization panel of Project Settings dialog box as follows:



Debug Control

Enable debug table generation

Produce debugging information in the operating system's native format (STABS, DWARF2 or DWARF).

Produce info in DWARF format

Produce debugging information in DWARF format.

Produce info in DWARF2 format

Produce debugging information in DWARF2 format.

Produce info in STABS format

Produce debugging information in STABS format.

Target uses the same floating point format

When running a cross-compiler, pretend that the target machine uses the same floating point format as the host machine.

Optimization Level

No Optimize(0)

Do not optimize.

General Optimize(1)

Optimize. Optimizing compilation takes somewhat more time and a lot more memory for a large function.

Most Optimize(2)

Optimize even more.

All Optimize(3)

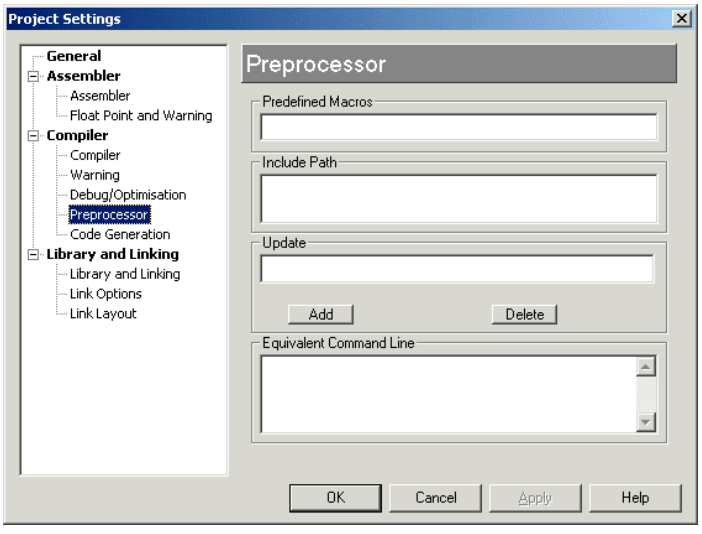
Optimize yet more.

4. Click on Apply button to save your changes temporarily.

Configuring the Preprocessor

Use the Project Settings to configure options for preprocessor:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Preprocessor in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Preprocessor panel of Project Settings dialog box as follows:



Predefine Macros

Define macros. It is an edit box and you can enter predefined macros separated by space.

For example, you can enter "Thumb HELLO GPF" to define three macros: Thumb, HELLO, and GPF. And "-DThumb -DHELLO -DGPF" thus appear in the Equivalent Command Line text box. Do not edit macros in Equivalent Command Line text box directly because Project Settings will check macros according to Predefined Macros edit box during saving.

Include Path

List the paths for current project which is used to search head file. You can add an include path by clicking on Add button. And you can remove a selected path item by clicking on Delete button.

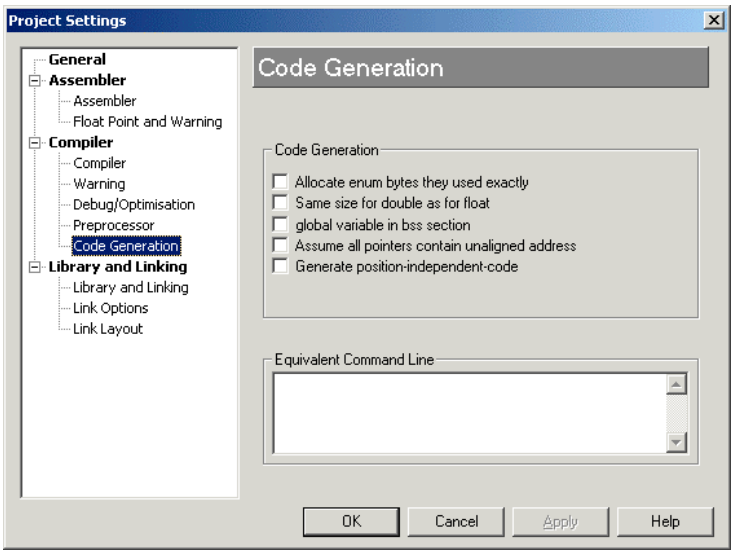
Once you select an item in the list in Include path, the content of the item will be displayed in the Update edit box

4. Click on Apply button to save your changes temporarily.

Configuring the Code Generation

Use the Project Settings to configure options for code generation:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Code Generation in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Code Generation panel of Project Settings dialog box as follows:



Code Generation

Allocate enum bytes they used exactly

Allocate enum bytes they used exactly.

Same size for double as for float

Same size for double as for float.

Global variable in bss section

Place the global variables in bss section.

Assume all pointers contain unaligned address

Assume all pointers contain unaligned address.

Generate position-independent-code

Generate position-independent-code.

4. Click on Apply button to save your changes temporarily.

Configuring Library and Linking Settings

This section describes the library and linking settings group of panels. These panels provide configuration options that are specific to the linker:

Use these panels to configure options for the GNU linker, including:

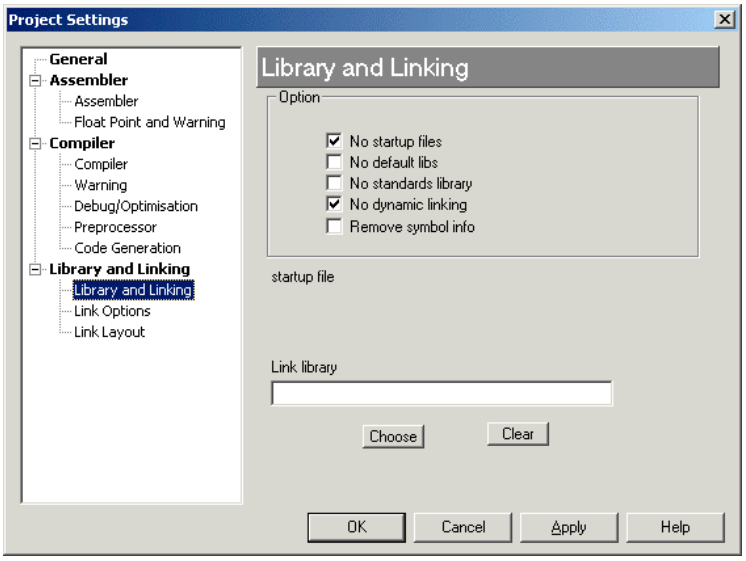
- Library and Linking
- Linking Options

- Linking Listings

Configuring the Library and Linking

Use the Project Settings to configure options for the library and linking:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Library and Linking in the Project Settings dialog panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Library and Linking panel of Project Settings dialog box as follows:



Option

No startup files

Do not use the standard system start-up files when linking. This is the default option because we supply the initial code ourselves.

No default libs

Do not use the standard system libraries when linking.

No standards library

Do not use the standard system startup files or libraries when linking. Same as "No startup files" + "No default libs".

No dynamic linking

On systems that support dynamic linking, this option prevents linking with the shared libraries. On other systems, this option is not in effect.

Remove symbol info

Remove all symbol table and relocation information from the executable.

Link library

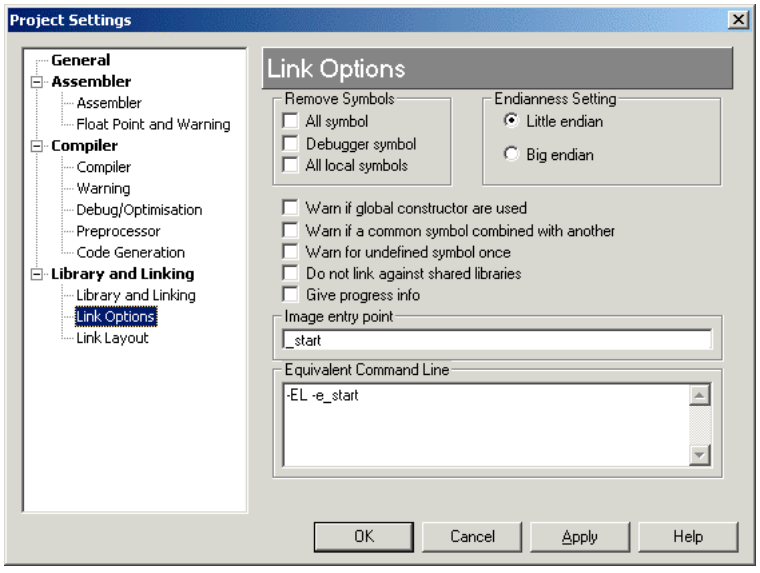
You can supply your own libraries to this project, you can enter them by clicking the "Choose" button. You can choose one library at one time. Please put your libraries in your project directory to ensure the libraries work correctly.

4. Click on Apply button to save your changes temporarily.

Configuring the Linking Options

Use the Project Settings to configure options for the linker:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Linking Options in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Linking Options panel of Project Settings dialog box as follows:



Remove Symbols

All Symbol

Remove all symbol table and relocation information from the executable.

Debugger symbol

Strip debugging symbols.

> ***NOTE***
>
> *You had better not set the above two options, otherwise there is no symbol information even after loading. You can not perform source level debugging.*

All local symbols

Discard all the local symbols.

Endianness Setting

Little endian

Little endian.

Big endian

Big endian.

Warn if global constructor are used

Warn if global constructors/destructors are seen.

Warn if a common symbol combined with another

Warn about duplicate common symbols.

Warn for undefined symbol once

Warn only once per undefined symbol.

Do not link against shared libraries

Do not link against shared libraries.

Give progress info

Output lots of information during link. If you set this check box, the progress information of linking will be displayed in build pane.

Image entry point

Entry of the whole program. For more detailed information, please refer to "Creating a project" section in this chapter carefully. It must be consistent with the entry in start-up code.

NOTE

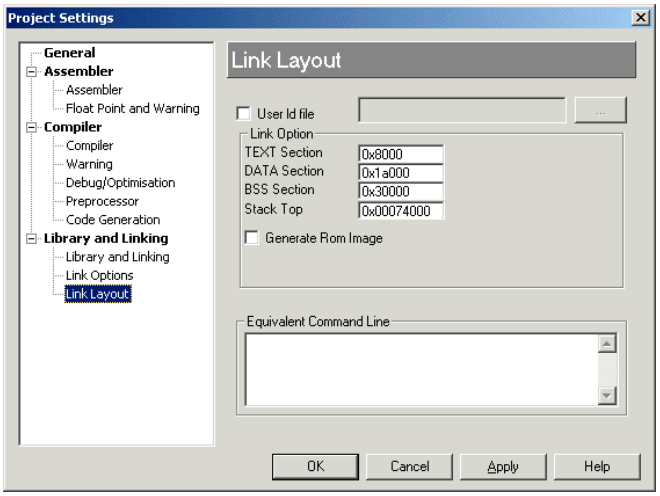
Do not enter a number, just a symbol in Image entry point option.

4. Click on Apply button to save your changes temporarily.

Configuring the Linking Listings

Use the Project Settings to configure options for the layout of the output file:

1. Display the Project Settings dialog box for the build target you want to configure.
2. Click on Linking Listings in the Project Settings dialog box panel list to display the configuration panel.
3. Choose and set the options as your requirement in the Linking Listings panel of Project Settings dialog box as follows:



Link Option

TEXT Section

Start address of .text section.

It must be a hexadecimal number, so the value in the edit box will be recognized as hexadecimal format.

DATA Section

Start address of .data section.

It must be hexadecimal number, so the value in the edit box will be recognized as hexadecimal format.

BSS Section

Start address of .bss section.

It must be hexadecimal number, so the value in the edit box will be recognized as hexadecimal format.

Be sure to allocate the three sections space carefully. Generally, .text section needs a larger space and .bss needs a smaller one.

Stack Top

Stack top of the stacks.

NOTE

You must prefix "0x" to indicate the address in the Stack Top box as hexadecimal format, otherwise it will be regarded as decimal.

Please refer to "Creating a new project" section in this chapter carefully. We strongly recommend you to set the value at the end address of RAM.

NOTE

All the addresses in the four options above must be word-aligned.

User ld file

If you hope your program a specific layout, you can supply your own link script file. The file is written by link command language.

There are three controls for this option: a check box, an edit box and a button. If you set the check box, the edit box and the button are enabled. You can enter the .ld file in the edit box directly or select it by clicking on the "..." button. That means you don't want to use the options about the linker in Project Setting dialog box.

NOTE

If you check "Use ld file" option, you can enter the .ld file Hitool created automatically in the current directory. Hitool will build the project together with .mak file it created in the same directory.

Here we give you a sample .ld file.

```
ENTRY( _start )
SEARCH_DIR( . )
__DYNAMIC = 0;

PROVIDE ( hardware_init_hook = 0);
PROVIDE ( software_init_hook = 0);

SECTIONS
{
        .text : {
                _ftext = . ;
                *( .init )
                eprol = . ;
                *( .text )
                *( .mips16.fn.* )
                *( .mips16.call.*)
                PROVIDE (__runtime_reloc_start = .);
                *(.rel.sdata)
                PROVIDE (__runtime_reloc_stop = .);
                *(.fini)
        }
        . = . ;
        .glue_7t : {

        }
        . = . ;
        .glue_7 : {

        }
        . = . ;
        .rodata : {
                *(.rodata)
                etext = .;
                _etext = .;
        }
```

```
. = ALIGN( 16 );
.data : {
        _fdata = . ;
        *( .data )
        CONSTRUCTORS
}
. = ALIGN( 16 );
. = . ;
.rdata : {
        *(.rdata)
}
. = ALIGN( 8 );
_gp = . + 0x8000;
__global = _gp;
.lit8 : {
        *(.lit8)
}
.lit4 : {
        *(.lit4)
}
.sdata : {
        *(.sdata)
}
. = ALIGN( 4 );
edata = .;
_edata = .;
_fbss = .;
.sbss : {
        *(.sbss)
        *(.scommon)
}
. = ALIGN( 4 );
.bss : {
        _bss_start = .;
        *(.bss)
        *(COMMON)
```

```
    }
    . = ALIGN( 4 );
    end = .;
    _end = .;
    _freeramStart = _end;
    _freeramEnd = _end + 0x1fffff;
    . = ALIGN( 4 );
    end1 = .;
    _stackBottom = 0x74000;

    .debug          0:{*(.debug)}
    .line           0:{*(.line)}

    .debug_srcinfo   0:{*(.debug_srcinfo)}
    .debug_sfnames   0:{*(.debug_sfnames)}

    .debug_aranges   0:{*(.debug_aranges)}
    .debug_pubnames  0:{*(.debug_pubnames)}

    .debug_info      0:{*(.debug_info)}
    .debug_abbrev    0:{*(.debug_abbrev)}
    .debug_line      0:{*(.debug_line)}
    .debug_frame     0:{*(.debug_frame)}
    .debug_str       0:{*(.debug_str)}
    .debug_loc       0:{*(.debug_loc)}
    .debug_macinfo   0:{*(.debug_macinfo)}

    .debug_weaknames  0:{*(.debug_weaknames)}
    .debug_funcnames  0:{*(.debug_funcnames)}
    .debug_typenames  0:{*(.debug_typenames)}
    .debug_varnames   0:{*(.debug_varnames)}
}
```

Click Apply to save your changes temporarily.

Using the Equivalent Command Line text box

The Equivalent Command Line text box is displayed at the bottom of configuration panel. You may find them in 9 panels, seven of which are useful:

1. Assembler
2. Float Point and Warning
3. Compiler
4. Warning
5. Debug/Optimization
6. Code Generation
7. Link Options

You can edit the options in the "Equivalent Command Line" edit box directly and Project Settings will compare the content in it and the options above in the panel. When you save options, Project Settings will generate options according to the options in the panel and your own input in "Equivalent Command Line".

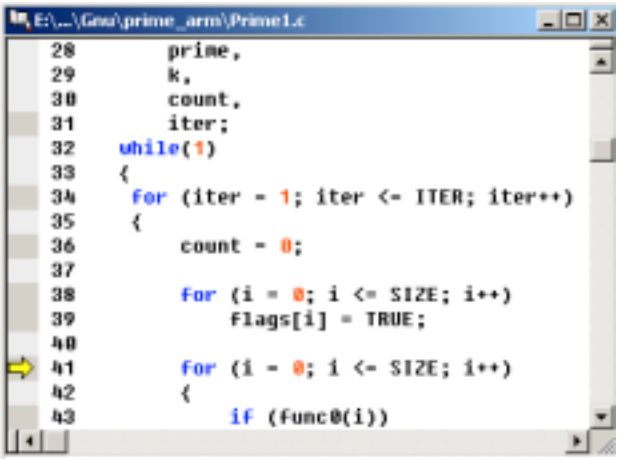
4 Viewing & Editing Source Code

Viewing Source Code

The Source Window

The Source Window combines a number of powerful options that make source level debugging quicker and more efficient.

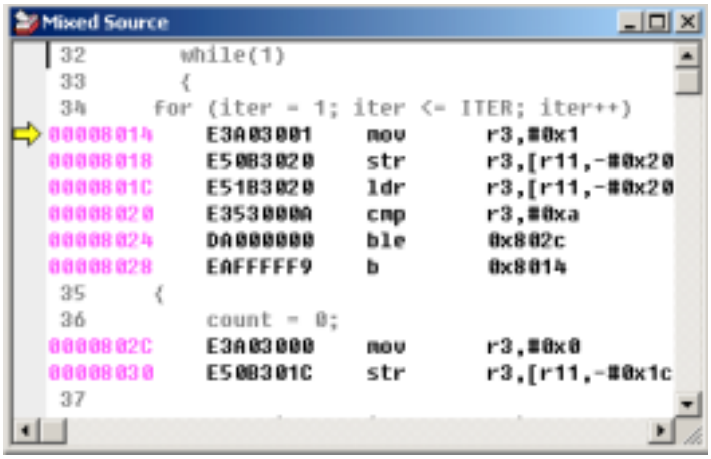
One of the main features of the Source Window is its drag and drop capabilities. Selected variables can also be dragged from the Source Window and dropped into Watch window directly.



The Source Window allows you to view source code in both high and low levels.

In the high level, line number source code can be viewed in Source Window.

In the low level, Assembly Language, Mixed Source can be viewed interleaved with source code in Mixed Source Window or Disassembly Window.



Furthermore, in the low level, Assembly Language, Mixed Source can be viewed interleaved with source code in Mixed Source Window or Disassembly Window in ARM Only, Thumb Only, Mixed ARM/Thumb mode. This feature is only available for ARM-Thumb interwork project.

Hitool offers two different ways to toggle among ARM Only, Thumb Only, Mixed ARM/Thumb mode.

- When either the Mixed Source or Disassembly Window is the active window, you can choose ARM Only, or Thumb Only, or Mixed ARM/Thumb from Disassembly mode item in the Option menu to display source code in corresponding mode.
- You can also right click on anywhere in the either the Mixed Source or Disassembly Window and select ARM Only, or Thumb Only, or Mixed ARM/Thumb menu item from the pop-up menu.

NOTE

In Assembly window, when you switch to Thumb mode, the source code may not display properly if they are compiled in ARM mode.

Hitool also offers two different ways to toggle between the high and low level screens.

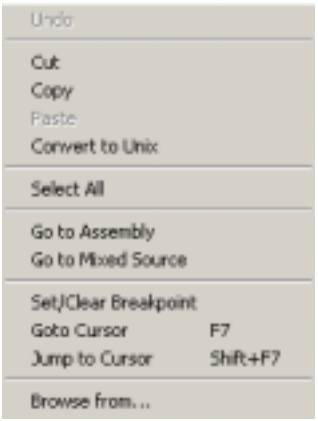
- When the Source Window is the active window, you can click on the Mixed, Disassembly icon in the Debug toolbar to open Mixed Source window and Disassembly window.
- You can also right click on anywhere in the Source Window and select the Go To Assembly, Go To Mixed Source menu item from the pop-up menu.

Features Include:

- **Drag and drop capabilities** - You can double click on to highlight a variable in the Source Window, then drag it into the Watch Window where it can be viewed directly then drop. Also you could select a pattern first, then drag and drop it in the source window to search the next.
- **Double clicking** - In the Source Window, you have the option of double clicking on a function to reposition to view the source code corresponding to that function.
- **Pop-up menu** - Right clicking anywhere in the Source Window invokes a pop-up menu containing added features.

Source Window Pop-Up Menus

When you right lick anywhere in the Source Window, and source is not selected, you will invoke the pop-up menu:



Convert to Unix - This item allows you to convert a text file from DOS format to Unix format. As DOS format, the line is ended up with "\0xa\0xd", Unix format ended with 0xa**.**

Go To Assembly - This item allows you to open a window displaying source in Disassembly language.

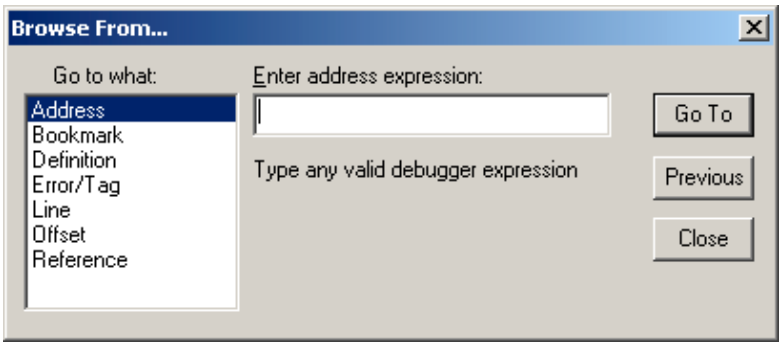
Go To Mixed Source – This item allows you to open a window displaying source in Mixed Disassembly and Source language.

Set / Clear Breakpoint - Selecting this menu item will allow you to set or delete an active software breakpoint. You may also simply click on the margin besides source line number where you wish to set a software breakpoint.

Go To Cursor - Selecting this menu item sets a temporary breakpoint. Hitool will send a GO command to the emulator. The breakpoint is automatically removed once it is hit.

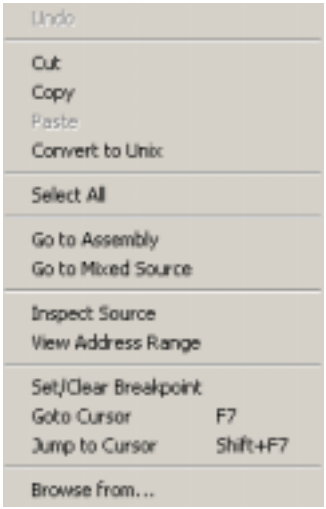
Jump To Cursor - Selecting this menu item sets the target's Program Counter to the address corresponding to the source line that you selected.

Browse From - Selecting this menu item invokes the following Browse From dialog box to enable you to locate to the right position quickly through Address, Bookmark, Definition, Error or Tag, Line, Offset and Reference.



Source Select Menu

To invoke the Source Select pop-up menu, right click once on a function in the Source Window to select it, or double-click, then right-click again.



Inspect Source – Relocate to the right position of source corresponding to the function.

View Address Range – Display the address range and the name of the function in the box.

Right click once on a variable in the Source Window to select it, or double-click, then right-click again to invoke the Source Select pop-up menu.

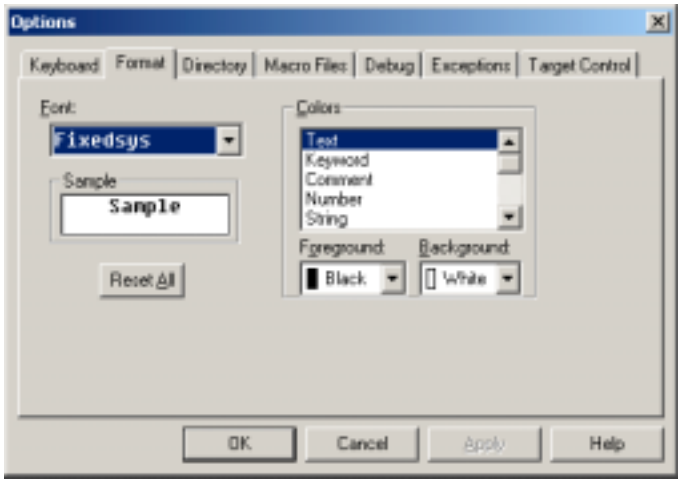
Watch Variable - Activate Watch Window and add the variable into it.

Inspect – Activate Inspect Window and add the variable into it.

Source Display Format

Hitool support the Built-in syntax-colored source display format.

To customize the Format, choose the Options menu item from Options menu to invoke the Options dialog box. You could define the display format of the source, such as font, colors of text and so on.



Source Paths

In order to debug, the debugger needs to know where the source modules are located. Thus, it is important to understand how to specify source paths and further, the process by which Hitool searches for source modules.

Locating Source Modules

If the debugger is unable to find one or more modules, it often uses source path information in the debug information contained in the executable file.

Then use the source specified in the project Source Files folder to diagnose the problem.

If the problem still exists, the debugger will automatically begin its search for modules using the source path(s) specified in the Options dialog box directory tab. You can also append multiple search paths or – if you Specifying Source Paths have many paths – you can list them and the debugger will use to search for modules.

If Hitool is unable to locate a source module via source path information specified in the Options dialog box directory tab, it will then popup a dialog box for user to specify the source file directly.

Specifying Source Path

Hitool's Options dialog box Directory tab permits to specify paths, as described in the following section.

Clicking the arrow in Search Order For will invoke the list box, from which you may choose Source Files then edit search paths, enter multiple paths to be searched, list multiple paths in paths list that Hitool will use to search.

Edit the Search Path(s) – Double click on the path then enter / edit the search path directly. When entering paths, be sure to separate paths with a semicolon.

Append Another Path to Search – Clicking on the button will invoke a dialog box from which you may choose the directory to enter the search path order.

If the path you have entered does not exist, a prompt will ask you if you wish to create this path.

Once all the paths are established, Hitool will look to the path list when attempting to locate modules and search in the order that the paths are entered, searching top to bottom.

Clear Path Information – Clears source path information.

Editing Source Code

Hitool IDE permits you to edit the source code freely. You can insert, delete the string from the active source window, save the changes you have made, select, copy, cut and paste the strings in the active source window. Here describes how to undo changes and search and replace text.

Undoing changes

The Hitool editor provides several methods to undo mistakes as you edit a file. The available methods are:

- Undoing the last edit
- Undoing multiple edits

Undoing the last edit

The **Undo** command reverses the effect of your last action.

To do this, choose the Undo Command on the **Edit** menu or from the shortcut menu of Source window. For example, you can select **Undo** to remove the text you just typed.

Undoing multiple edits

You can undo multiple edits when selecting Undo command multiple times.

To do this, select **Undo** from the **Edit** menu or from the shortcut menu of Source window multiple times to undo your previous edits.

For example, if you cut a word, paste it, then type some text, you can reverse all three actions by choosing **Undo** three times. The first undo removes the text you typed, the second unpastes the text you pasted, and the third uncuts the text you cut to restore the text to its original condition.

The maximum times of the actions you can undo is defined in Max Undo Len option in Debug tab of Options dialog box.

Searching and Replacing Text

This section describes how to use the Hitool IDE search and replace functions.

The search and replace facilities in the Hitool IDE enable you to search and replace from either the current editor window or other text files. You can search for and replace text in a single file, in every file in a project, or in any combination of files.

The Find dialog and Replace dialog, shown as below, provide comprehensive search and replace facilities.

The Find dialog box enables you to search for text patterns in the active editor window. When you find the text you are searching for, you can change it or look for the next occurrence.

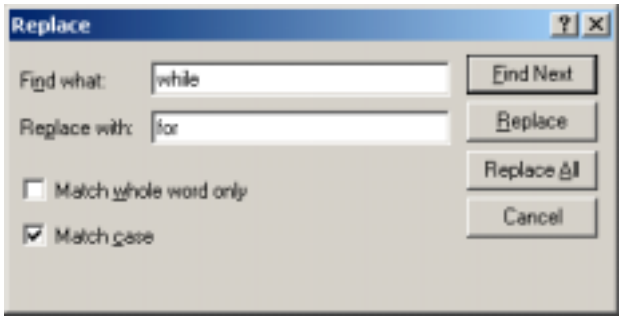
Find dialog box

If you are looking for repeated instances of the same string, do one of the following:

1. Click on Find Next button in Find Dialog box.
2. Choose Find Next from Edit menu.
3. Click on Find Next button on Edit Toolbar.

You can use the Replace dialog to perform find replace operations for:

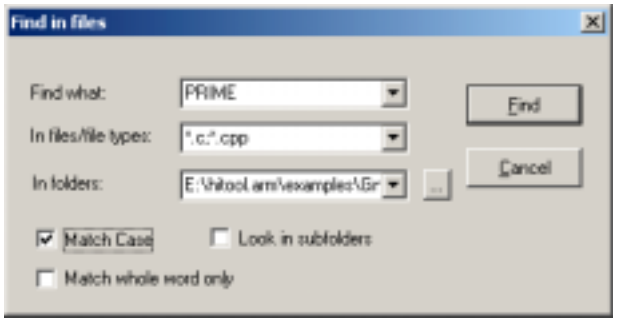
- text in a single file
- text in arbitrary files that are not part of your current project.



Replace dialog box

You can use text strings, text substrings, and pattern matching in find and replace operations.

In addition, you can use the following Find In Files dialog box to perform batch search and display the results of a find operation in Find In Files pane in Output window.



Find In Files dialog box

The Hitool IDE Find In Files dialog enables you to search multiple files for the occurrence of text strings. The Find In Files dialog box enables you to specify searches through:

- Source files in a project
- Header files in the project

- Any additional files you specify.

Closing the Source Window

To close the Source Window, use the window’s control handle located in the upper left-hand corner of the window.

- Click on the control handle and select the Close menu item in the pop-up window control menu.
- Double-click on the window's control handle.

5 Controlling Execution

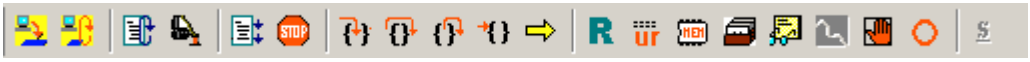
Controlling Execution

Hitool offers a number of options for controlling execution in both high and low level modes, giving you greater control over your source code.

To make controlling execution even quicker, Hitool incorporates a number of shortcut keys. The keys are listed below and can be found on the Debug menu.

Debug Toolbar

The debug toolbar allows you to control execution quickly and efficiently.



Debug Menu

The debug menu gives ability to control execution quickly and efficiently.

Key	Function
F2	Halt
F5	Go
F7	Go to Cursor
F8	Step Into
F9	Set SW Breakpoint at the line of cursor
F10	Step Over
Alt+F5	Free Run

The following lists the description of the functions controlling execution:

		SOURCE LINES (High Level mode)	INSTRUCTION (Low Level mode)
	Reset	Resets the target , and Set emulator PC to the specified address directly if the symbol has been loaded	Resets the target , and Set emulator PC to the start address directly if the symbol has been loaded
	Restart	Similar to Reset, but it will go from the start address to the C main entry if it is specified. You can specify the related option in Debug tab of Options dialog box.	Similar to Reset, but it will go from the start address to the C main entry if it is specified. You can specify the related option in Debug tab of Options dialog box.
	Go	Runs the target code.	Runs the target code.
	Free Run	Runs the target code, ignoring the	Runs the target code, ignoring

		software breakpoint.	the software breakpoint.
	Halt	Stops the target code.	Stops the target code.
	Step Into	Steps one source line. If the line contains a function call, the debugger steps into the function call.	Step one instruction.
	Step Over	Steps on source line. If the line contains a function call, the debugger uses a temporary breakpoint to step over that function, going to the next logical source line in the current function.	Step one instruction. If the instruction is a BL, the debugger uses a temporary breakpoint to step over that instruction, moving to the next instruction.
	Step Out (Only available for C Code)	Single steps out of the current function to a newly called function, or the return function, whichever occurs first.	Single steps out of the current subroutine to a newly called subroutine, or the return subroutine whichever occurs first.
	Step Into Continuously	Continuously single steps on source line.	Continuously single steps on one instruction ignoring breakpoint.
	Step Over Continuously	Continuously single steps over source line.	Continuously single steps on one instruction even if the instruction is a BL ignoring breakpoint.
	Go To Cursor	Runs the target code to the address corresponding to the source line which the cursor points to.	Runs the target code to the instruction which the cursor points to.
	Show Execution Context	Activates Source window and displays from the source line corresponding to current PC.	Activates Disassembly Window and displays from the instruction corresponding to current PC.

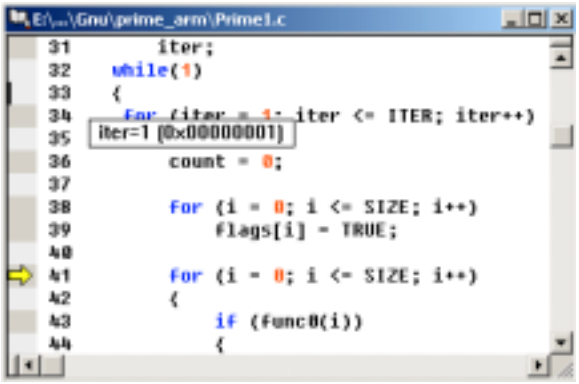
6 Displaying Data, Memories, Registers

Displaying Data

Hitool gives you three different ways of displaying data. You can view data through the datatip in the Source Window, an inspect, or for even greater control over your data, you can use Hitool's powerful Watch Window.

Datatip

One of the easiest ways to view data is through the datatip. To view the datatip, place the mouse on the desired variable in the Source window wait a moment until the datatip appears, giving the information.

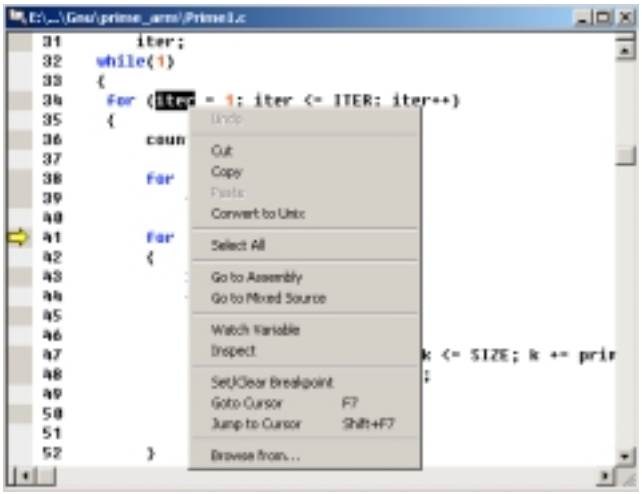


Inspect

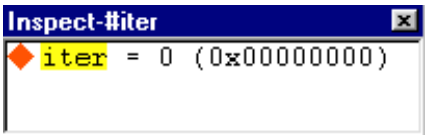
Another simple and convenient way to view data is through an Inspect. To view an Inspect, select a variable in the Source Window and right-click.

Inspect window is always on top, but only view a variable. You can open multiple Inspect windows.

Highlight the variable in the Source window and right click on. From the pop up menu, select the Inspect item.



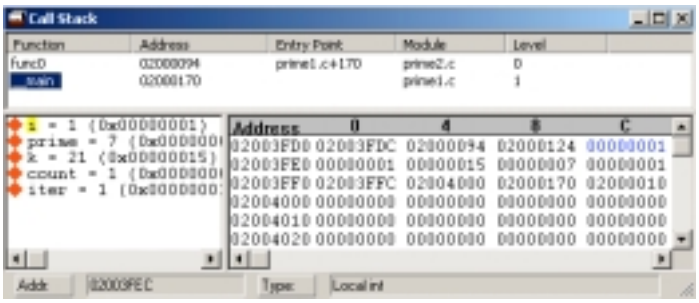
From the Inspect Window, you may modify the variable as wished.



Stack Window

You can open the Stack Window by doing any of the following:

- Locating the Debug toolbar and clicking on the Stack Window icon.
- Clicking on the Debug Window menu item from the View Menu. Then choose Stack item.



Stack Window

The Stack window contains two panes. You may resize the two panes by pulling the cutting bar when the mouse shape changes to a clipper. When one is enlarged, the other is reduced correspondingly.

Upper Pane

The upper pane displays the function calling sequence in the call stack up to the current function.

In the upper pane, you can inspect source, set breakpoint directly and run by choosing the item from the pop up menu.

All of the columns can be resized by drag the column margins when it becomes clipper.

Lower Pane

The lower pane consists of two sections. The left one shows parameters and local variables of the highlighted function in the upper pane. It can be seen as a miniature of the Watch window. Not only does it provide most of the features of the Watch window, but also it even has the same look as the Watch window. You can modify the value of highlighted variables.

The right one shows the memory content corresponding to the function stack pointer. All levels of functions in the upper pane can

be located in Memory window in different colors. Highlight the function, the Memory window will locate to the right address of the stack pointer of the function indicating and the content is displayed in corresponding color. You can scroll the vertical bar of Memory window to locate to the next or previous content.

> ***NOTE***
>
> *The shortcut menu of the upper pane will be activated only when you highlight the items in them.*

Displaying Memories, Registers

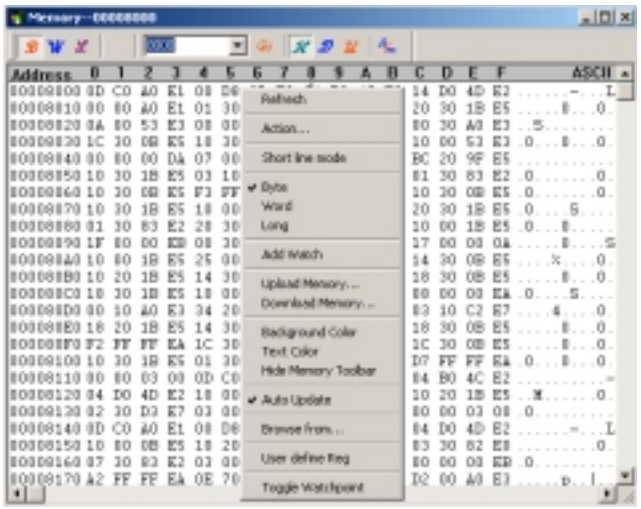
Hitool gives you convenient ways of displaying memories, registers. You can view memories through the Memory Window. For even greater control over registers, you can use Hitool's powerful Register Window and Peripheral Window.

Display Memories

Display General Memories

You can open the Memory Window by doing any of the following:

- Locating the Debug Toolbar and clicking on the Memory Window icon.
- Clicking on the Debug Window menu item from the View Menu. Then choose Memory item.

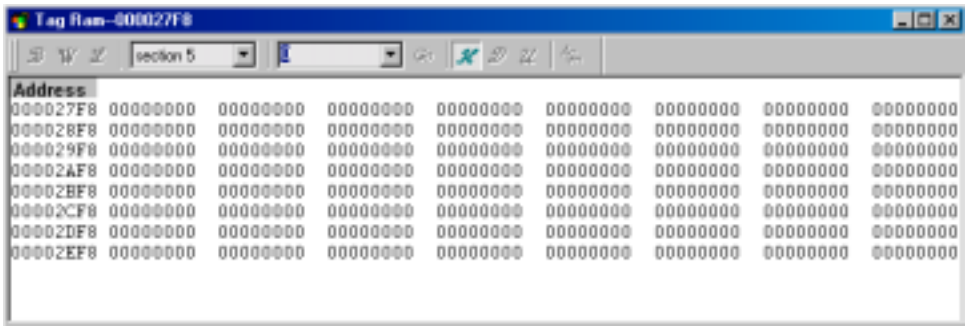


Hitool offers a number of different options for displaying memory in the Memory Window.

- **Memory Toolbar** – Click on the icon on Memory toolbar to display memory in different format.
- **Pop-up Menu** – Right-click in the Memory Window to invoke the pop-up menu. Select the menu item to display memory in different format.

Display PA TAG RAM

920T implements PA TAG RAM to perform write-back from Data Cache. Hitool provides an UI to display it to make the user aware. This window is read only.

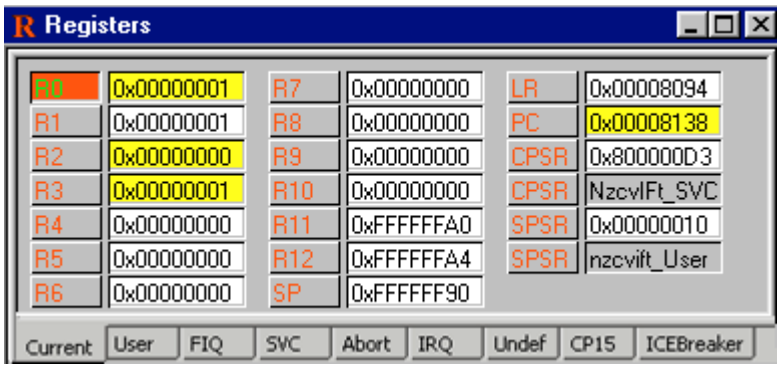


Display Registers

Display General Registers

You can open the Register Window by doing any of the following:

- Locating the Debug toolbar and clicking on the Register Window icon.
- Clicking on the Debug Window menu item from the View Menu. Then choose Register item.



Float View – Choose Float View menu item from the pop-up menu to display the general register value in selected tab during emulation which is always on the top.

Add Watch - Choose Add Watch menu item from the pop-up menu to add the highlighted general register into Watch Window.

Indirect Through Register - Choose Indirect Through Register item from the pop-up menu to activate Memory Window and locate to the right address the highlighted special function register indicating.

CP15 Register

CP15 registers are used in conjunction with page table descriptor stored in memory to determine the operation of the MMU. So the first thing is accessing the CP15 registers.

To access the CP15 registers, two internal variables, CP15_cache_selected and CP15_current_memory_area, are used. Because standard CP15 registers are used for more than one function. These two variables will determine the exact meaning of each register accessed.

• CP15_current_memory_area

Normally this variable indicates I-Cache (1) or D-Cache (0). This is only valid for Harvard Architecture processors. The possible value is:

0: Data Cache;

1: Instruction Cache

2: tightly coupled instruction memory

3: tightly coupled data memory

- CP15_current_memory_area

Just for 740T and 940T. Select the memory area to be used with accesses to register 6.

920T CP15 Register Mapping

• c0	Read-only	
	CP15_cache_selected=0	ID Code
	CP15_cache_selected=1	Cache Configuration
• c1	Read/Write	Control
• c2	Read/Write	TTB
• c3	Read/Write	DAC(Domain Access)
• c5	Read/Write	
	CP15_cache_selected=0	FSR(Data Abort)
	CP15_cache_selected=1	PFSR(Prefetch Abort)
• c6	Read/Write	Fault Address
• c7	Write Only	Cache Operations

Bit[2:0] of the data value written in conjunction with the value of CP15_cache_selected decide on the function performed:

CP15_cache_selected	bit 2	bit 1	bit 0	Function
1	0	0	0	Invalidate ICache & DCache
1	0	0	1	Invalidate ICache
1	0	1	0	Invalidate I single entry (VA)
1	0	1	1	Prefetch ICache Line (VA)
0	0	0	0	Invalidate DCache
0	0	0	1	Invalidate D single entry
0	0	1	0	Clean D single entry (VA)
0	0	1	1	Clean & Invakidate D single entry (VA)
0	1	0	0	Clean D single entry (index)
0	1	0	1	Clean & Invakidate D single entry (index)
0	1	1	0	Drain Write buffer

- c8 Write Only TLB Operations

Bit[1:0] of the data value written in conjunction with the value of CP15_cache_selected decide on the function performed:

CP15_cache_selected	bit 1	bit 0	Function
1	0	0	Invalidate ITLB & DTLB
1	0	1	Invalidate ITLB
1	1	0	Invalidate ITLB single entry
0	0	0	Invalidate DTLB
0	0	1	Invalidate DTLB single entry

- c9 Read/Write Cache lockdown
  - CP15_cache_selected=0 DCache lockdown base
  - CP15_cache_selected=1 ICache lockdown base
- c10 Read/Write TLB lockdown
  - CP15_cache_selected=0 Data TLB
  - CP15_cache_selected=1 Instruction TLB
- c13 Read/Write Processor ID

• c15 Read/Write Test and Debug

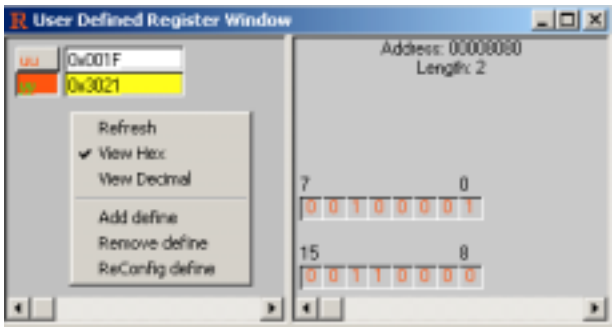
NOTE

1. *ICEBreaker pane isn't supported in simulator mode, so the registers value in ICEBreaker pane are false in simulation mode.*
2. *If banked registers aren't in current mode, the value of banked registers maybe incorrect.*
3. *The first seven panes are always available. The last six pane maybe not true and unwrite-able for Angel or Jeeni Protocol.*

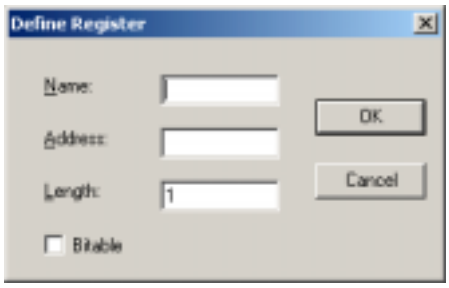
Display User Defined Registers

You can open the User Defined Register Window by doing any of the following:

- Locating the Debug toolbar and clicking on the Defined Reg Window icon.
- Clicking on the Debug Window menu item from the View Menu. Then choose Defined Reg item.



Add Define – Choose Add Define menu item from the pop-up menu to display the Define Register dialog box to define the register.



User can define some registers to monitor the corresponding memory. All the registers will be recorded in the defreg.cfg file in Hitool for ARM installation directory in the following format:

Name StartAddr Length Bitable

Name can be up to 8 characters, mixed with alphabetic character and numeric character, but the first character must be an alphabetic character.

StartAddr can be numeric or symbolic.

Length is limited. Usually the length can be from 1 to 4.

NOTE

Start address and end address should be in the same word range.

Bitable (0|1) indicates whether the register can be bit-expanded.

Remove Define - Choose Remove Define item from the pop-up menu to remove the highlighted user defined register from the User Defined Register Window.

Reconfig Define – Choose Reconfig Define menu item from the pop-up menu to display the Define Register dialog box to reconfig the register.

Display Peripheral Registers

You can open the Peripheral Window by doing any of the following:

- Locating the Debug toolbar and clicking on the Peripheral Window icon.
- Clicking on the Debug Window menu item from the View Menu. Then choose Peripheral item.

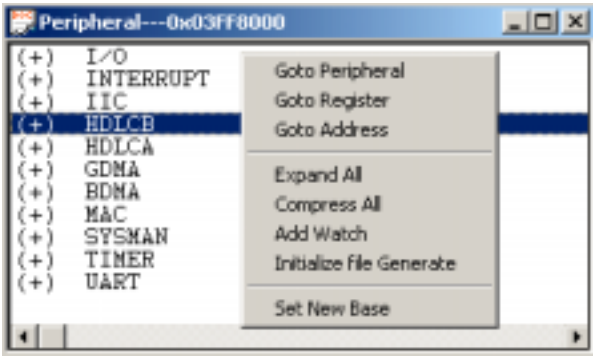
Peripheral Window Display

Peripheral window is displayed in compressed format by default.

Compressed Registers Display

The columns in the Peripheral window are (from left to right):

- A (+) symbol.
- Memory-mapped Control Registers group mnemonic.



Peripheral Window in compressed format

Click on a (+) symbol in the left column of the registers associated with a group.

Click on an indented (+) symbol to view the bits associated with a register.

Goto Peripheral – Choose Goto Peripheral menu item from the pop-up menu to invoke Go To Peripheral dialog box, then fill in the peripheral name you want to locate to.

Goto Register – Choose Goto Register menu item from the pop-up menu to invoke Go To Register dialog box, then fill in the register name you want to locate to.

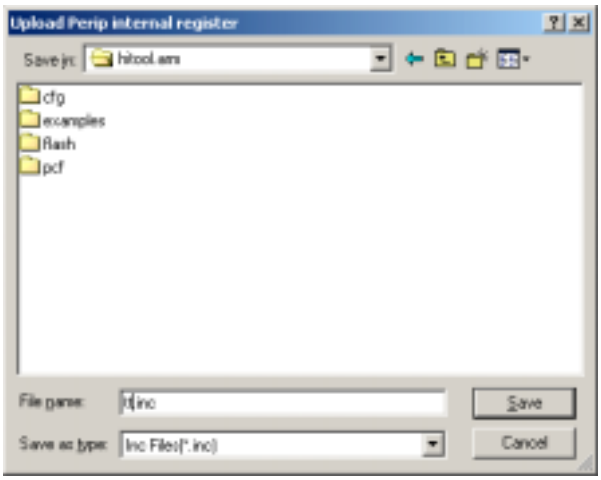
Goto Address – Choose Goto Address menu item from the pop-up menu to invoke Go To Address dialog box, then fill in the offset of the register you want to locate to.

Expand All – Choose Expand All menu item from the pop-up menu to expand all the registers.

Compress All – Choose Compress All menu item from the pop-up menu to compress all the registers.

Add Watch - Choose Add Watch menu item from the pop-up menu to add the highlighted register into Watch Window.

Initialize file Generate - Choose Initialize file Generate menu item from the pop-up menu, in the following pop-up Upload Perip internal register dialog box you can save current registers value into an include file for initialization.



Set New Base - For Samsung KS32C50100, SYSCFG (a perip register) bit16~bit25 indicate the start address of the perip registers. If the user changes the SYSCFG content, all the perip registers will be moved. Hitool maybe see this change, but in some cases, for example, the user application code change it, Hitool will ask the user set the perip registers start address manually.

Selecting Set New Base, a Set Peripheral Base Address dialog box will pop up, in which you can enter a new address.

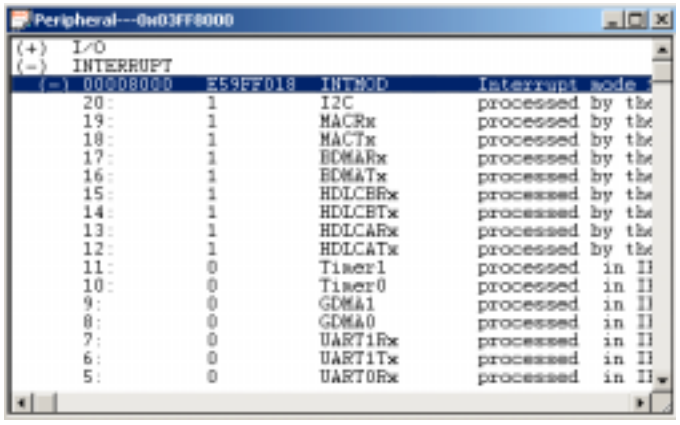
NOTE

Expanded format allows you to view the individual bit fields of the register. If a register contains only one field, then it is not expandable.

Expanded Registers Display

The columns in the Peripheral window are (from left to right):

- A (+) or (-) symbol.
- The relative address of the register or register field bit.
- Register value or the register field value.
- Register mnemonic or field mnemonic.
- Register description or field description.


Peripheral---0x03FF8000
(+) I/O
(-) INTERRUPT
(-) 00008000 E59FF018 INTMOD Interrupt mode
20: 1 I2C processed by th
19: 1 MACRx processed by th
18: 1 MACTx processed by th
17: 1 BDMARx processed by th
16: 1 BDMATx processed by th
15: 1 HDLCBRx processed by th
14: 1 HDLCBTx processed by th
13: 1 HDLCARx processed by th
12: 1 HDLCATx processed by th
11: 0 Timer1 processed in II
10: 0 Timer0 processed in II
9: 0 GDMA1 processed in II
8: 0 GDMA0 processed in II
7: 0 UART1Rx processed in II
6: 0 UART1Tx processed in II
5: 0 UART0Rx processed in II

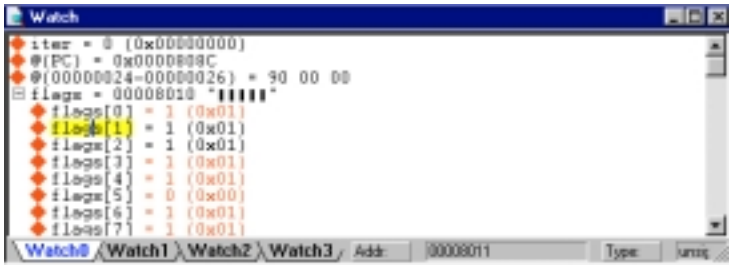
Peripheral Window in expanded format

Watch Window

The most powerful way of displaying and controlling data, memories and registers is through the Watch Window.

The Watch Window allows you to continuously monitor or inspect local and global variables, memories and registers in your application. Every time the target stops, the Watch Window is updated with the variable's, memory's and register's most current value.

The Watch Window provides an outline format for watching data, memories and registers. You can expand or contract with just a simple click.



Opening the Watch Window

You can open the Watch Window by doing any of the following:

- Locating the Debug toolbar and clicking on the Watch Window icon.

- Clicking on the Debug Window menu item from the View Menu. Then choose Variable item.

Placing Data into the Watch Window

With the Watch Window open, there are a number of ways to place information into the window so that it can be viewed or modified.

The Source Window

The first step in placing data into the Watch Window is to select a variable, which can be done a number of ways:

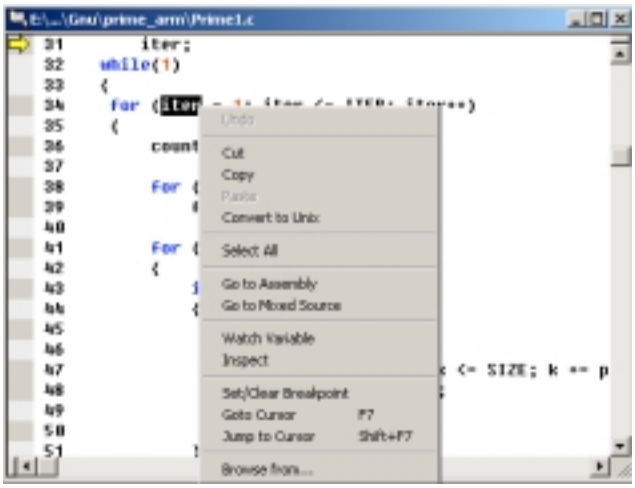
- Double-click on a variable.
- Right-click once on a variable.
- Click in front of the variable. While holding the button down, you can drag the mouse across the variable to select it.

After selecting a variable in the Source Window, there are a number of different ways to place it into the Watch Window.

One method is to click on the variable after it has been selected. When the folder icon appears, you can drag it directly into the Watch Window.

Another method for moving data is to right-click on the selected variable to invoke the pop-up menu.

From the menu, select the Watch Variable item.



Project Window Symbols Pane

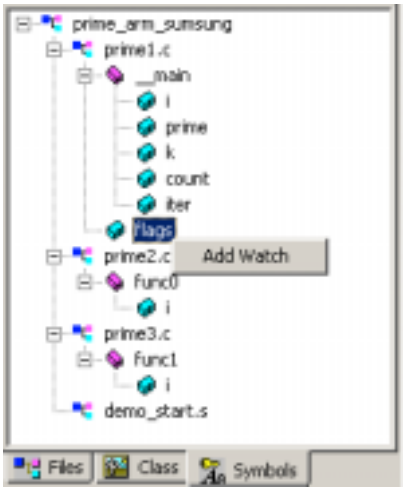
Hitool also allows you to select and place either local or global variables into the Watch Window from the Project Window Symbols Pane.

Global Variables

To place a global variable into the Watch Window, you can either drag the variable directly into the Watch window, or use the pop-up menu.

To drag the variable, click once to select the variable. When the folder icon appears, keep the mouse button depressed and drag the variable into the Watch Window then drop.

You can also select the variable and right-click to invoke the pop-up menu. From the menu, select the Add Watch item.

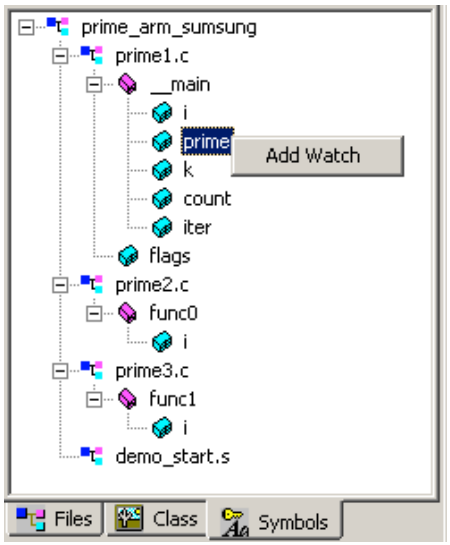


Local Variables

To place a local variable into the Watch Window, click on the + next to the Function heading to open the Function outline. Then, click on the desired function to display a list of the local variables within the function.

With the Locals list open, you can drag the variable into the Watch Window just as you would a global variable.

You also have the option of right-clicking on a local variable and selecting the Add Watch item from the pop-up menu.

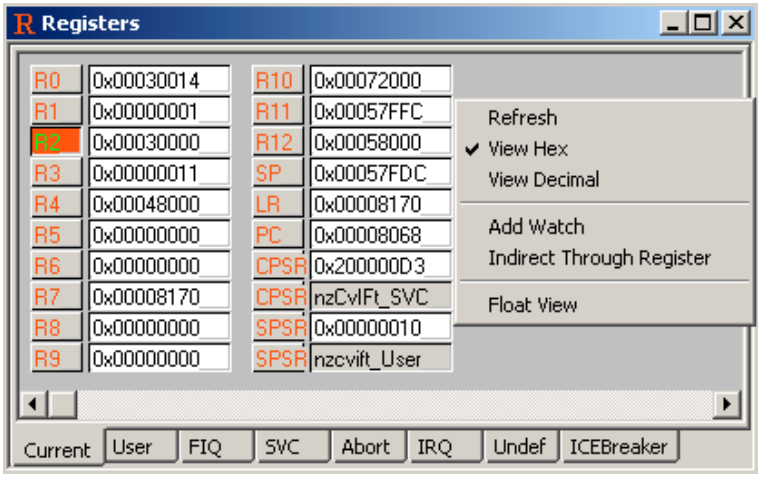


Placing Registers, Memories into the Watch Window

Registers Window

With the register window open, you can click once to select a register. You can then drag the register directly into the Watch Window then drop.

Under the register name, value, or other space, you also have the option of selecting the register and right-clicking on to invoke the pop-up menu. Select the Add Watch menu item to place the register into the Watch Window.



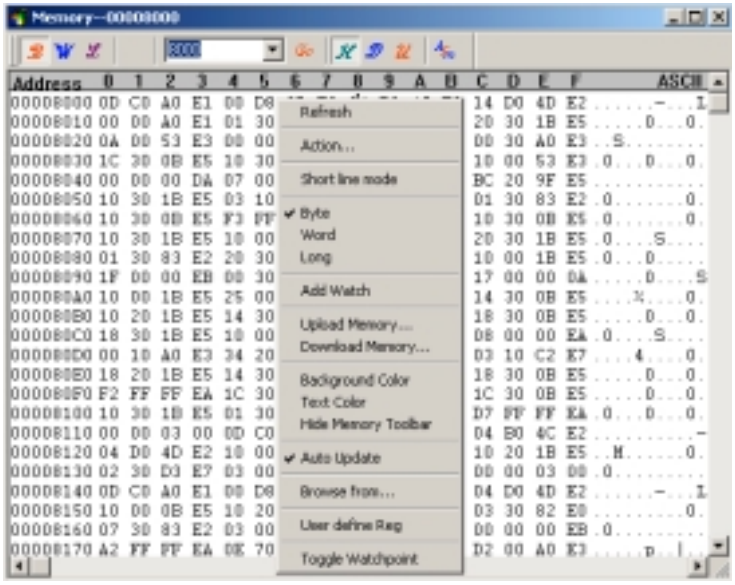
Peripheral Window

With the Peripheral Window open, you can click on to select a register.

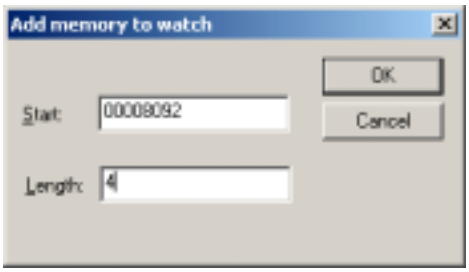
Under the register name, value, or other space, you also have the option of selecting the register and right-clicking on to invoke the pop-up menu. Select the Add Watch Window item to place the register in the Watch Window.

Memory Window

With the Memory window open, you have the option of right-clicking to invoke this pop-up menu:



Select the Add Watch item to invoke the following Add memory to Watch dialog box.

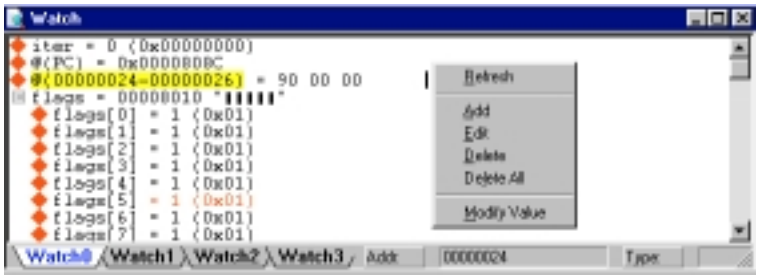


In the dialog box the start address is "8092", then fill in the length which is an integer less than sixteen.

Click on OK button to close the dialog box or click on Cancel button to cancel.

Watch Window

You can retrieve data directly from the Watch Window by right-clicking to invoke the pop-up menu. Select the Add Item.



Once this item is selected, a blank box will appear, allowing you to enter a variable, register, memory.

Variables - To add a variable in the Watch Window, type in the variable you wish to view.

Memories - To view memory of specified address, type in the start address and end address you wish to view. Be sure that the format of the item you type must be the same as the format of its kind in the Watch window.

Registers - Placing a register into the Watch Window requires you type the register you wish to view. Be sure that the format of the item you type must be the same as the format of its kind in the Watch window.

Viewing Data, Registers, Memories in the Watch Window

The Watch Window allows you to continuously monitor, or inspect, local, and global variables, registers, memories in your application. Every time the target stops, the Watch Window is updated with the most current value of variables, registers and memories.

The Watch Window provides an outline format for watching datum, registers memories. You can expand, contract, or add information with a simple click.

Watch Window Tabs

There are four tabs running across the bottom of the Watch Window:

Each of the watch tabs allows you to maintain separate data that is continuously updated. The watch tabs also enable you to move between the information on each tab with the click of a mouse.

Right-click anywhere in the Watch Window and a pop-up menu appears. A description of main menu item found on this menu follows:

Add – Choose the Add item to add a new item. You can also type to add the item you wish to view.

Edit - Choose the Edit item to delete the selected item and add a new item.

Delete - Choose the Delete item to remove the selected item.

Delete All - Choose the Delete All item to remove all items on the selected tab.

Modify Value - Choose the Modify Value item to set the selected item's value. Type in a new value.

Changing Data

In addition to offering a number of options for viewing data, Hitool provides a number of ways to change data.

- You can also change data in the Watch Window by selecting a data, and right-clicking to invoke the pop up menu. Select the Modify Value item and type a new value in the box.
- To change data in the Stack Window, select a data in the lower pane, and right-clicking to invoke the pop up menu. Select the Modify Value item and type a new value in the box.
- Data can be deleted in the Watch Window by selecting the variable and pressing the <Delete> key.
- Data can also be deleted in the Watch Window by selecting a variable and right-clicking to invoke the pop-up menu. Choose Delete or Delete All item from the menu to delete the data.

Changing Registers, Memories

Register Window

When a register value is changed (i.e., when it has a new value from when it last halted), the register value is displayed with yellow background. Register window display is updated when emulation halts.

When click on Reset button on the debug toolbar to reset and reinitialize the target processor, the following happens:

1. The processor reset pin is asserted.

2. All windows are updated with the following consequences:

The Stack Window display is invalid because the stack is reset.

PC counter points to the start of code.

Modify General Register

You can modify general register value directly. To modify general registers, do the following:

1. Highlight the general register which you want to modify, double click on the value panel of register.

2. Fill in the new value in the pane then press Enter key.

The modified general register value is displayed with yellow background.

Peripheral Window

When choose Reset command of debug menu in menu bar or Reset button in Debug Toolbar to reset and reinitialize the target processor, the following happens:

1. The processor reset pin is asserted.

2. All windows are updated with the following consequences:

The Stack window display is invalid because the stack is reset.

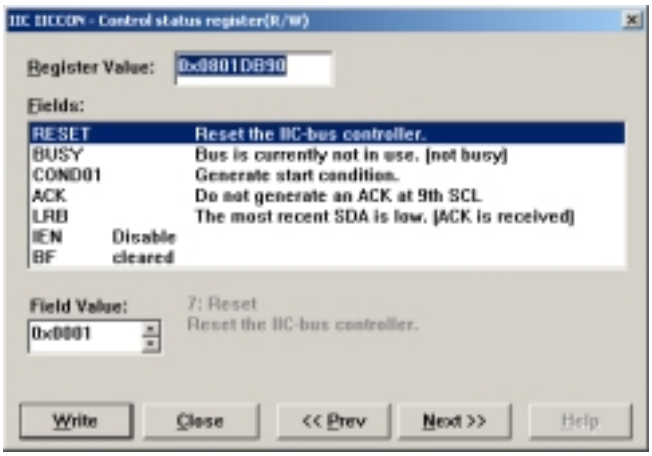
PC counter points to the start of code.

Edit a Memory-mapped Control Register in the Peripheral Window

Double click on anywhere on a register line to bring out the dialog box. This dialog box enables you to edit the register value or the individual register fields.

If you double click on some field of a register, the dialog box displays the dialog box that field is highlighted. For example,

1. Expand the IIC group and expand the Control Status Register.
2. Double click on any fields of the register to bring out the following dialog box.



3. Change the value with the Field Value spin box or fill in the value directly.
4. Click on Write button to change the register. The dialog box stays open, enabling you to change other register fields.
5. Click on Close button to close the dialog box.

-or-

1. To simplify this example, compress the register display and highlight the IIC group you want to edit. Then double click on IICCON to bring out the dialog box.

2. Click on Next>> button. Note that the highlight in the Peripheral window moves to the next IICBUF group, the corresponding register value is displayed in the dialog box. Click on <<Prev button to return to the previous group.

Memory Window

Type over the memory contents, you can directly modify the memory contents.

7 Breakpoints&Watchpoints

Breakpoints&Watchpoints

Hitool offers a number of different ways to set breakpoints and watchpoints, making your debugging quick and easy.

Software Breakpoints can be set from:

- The Source Window
- The Breakpoint dialog box
- The Breakpoints Window
- The Project Window Symbols Pane
- The Stack Window
- The Shell command Window

Watchpoints can be set from:

- The Source Window
- The Memory Window
- The Watchpoint dialog box
- The Breakpoints Window
- The Shell command Window

Software Breakpoints

Software breakpoints are used by most source-level debuggers as the standard way to halt the processor at a code address.

Hitool also allows you to set complex software breakpoints. Complex Breakpoints allow you to set an if / then condition for the breakpoint, and thus narrow the focus of the break. Hitool uses the same instruction swapping scheme to break the target, with the break contingent on the validity of the if / then condition.

Since software breakpoints use the aforementioned instruction swapping scheme, code must be running within RAM.

Watchpoints

Watchpoints occur on data accesses or instruction prefetch.

Wiggler series implement breakpoint and watchpoint via Watchpoint unit (ICE Breaker). Breakpoint will occur watchpoint unit 1, and the other watchpoint unit can implement instruction prefetch or data access watchpoint.

A watchpoint is always taken, but the core may not enter debug state immediately. In all cases, the current instruction completes. If the current instruction is a multiword load or store (an LDM or STM), many cycles may elapse before the watchpoint is taken.

When a watchpoint occurs, the current instruction completes, and all changes to the core state are made (load data is written into the destination registers, and base write-back occurs).

If a watchpoint occurs when an exception is pending, the core enters debug state in the same mode as the exception.

Each watchpoint unit can be configured to be either a watchpoint (monitoring data accesses) or a breakpoint (monitoring instruction fetches). Watchpoints and breakpoints can be data-dependent.

Angel series watchpoint is just data-changed watchpoint, that is, they are not activated if the data point is read or written at the same data as the one currently in memory. Therefore the Angel series watchpoint dialog is similar to the breakpoint dialog box.

Watchpoints are taken when the data being watchpointed has changed. When this happens, the PC is updated to point to the instruction following the one that caused the watchpoint to be taken. Then value of the watchpointed data is therefore the new data.

NOTE

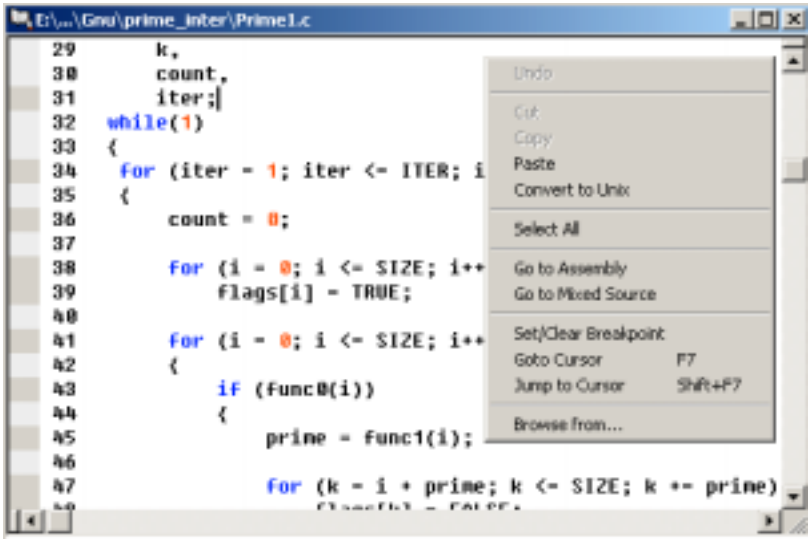
The functions about Watchpoint, such as Watchpoint dialog box, Watchpoint shell command and so on, are not available in simulator.

Setting Breakpoints

> ***NOTE***
>
> *Symbols must be loaded before you can define breakpoints on line numbers or functions.*

Source Window

One of the easiest and most convenient ways to set breakpoints in Hitool is through the Source Window. Click on any line number from within the Source Window to invoke the pop-up menu as shown below:



Options for setting breakpoints include:

- To set a software breakpoint you can also simply click on any source line number in the Source Window or low-level address in the Mixed Source Window and Disassembly Window.

Bull's-Eye

Breakpoint Symbol

Point the mouse to the left of a source line, Hitool displays a breakpoint (bull's-eye) cursor. Then click on the source line to set a breakpoint. A breakpoint set symbol (red dot) appears to the left margin of the source line.

NOTE

If you try to set a breakpoint on a non-executable statement, you will get no response.

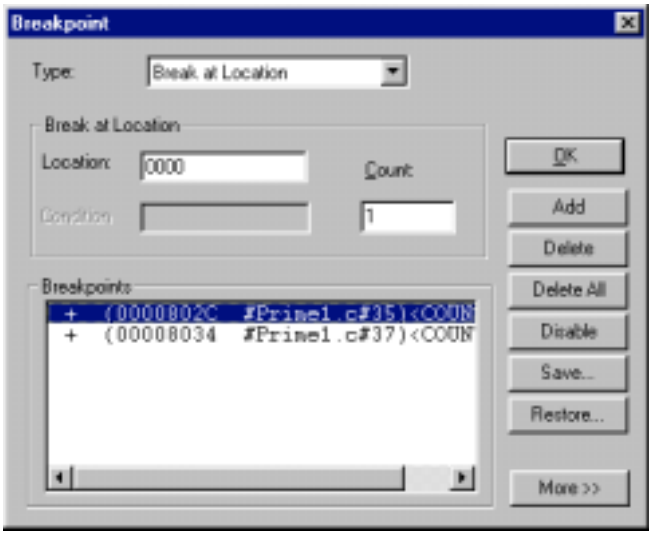
Setting Breakpoint through Breakpoint dialog box

You can set specific software breakpoint by accessing the Breakpoint dialog box as follows:

1. Choose Breakpoint command from the Debug menu.

   Then set the breakpoint in the Breakpoint dialog box as illustrated below.

2. Select the breakpoint type "**Break at Location**" from the **Type** combo box.
3. Fill the address 802C in the **Location** field, and **Count** field is 1 by default.



NOTE

You can enter a numeric or symbolic address. Numeric addresses are hexadecimal addresses such as 802C. An example of a symbolic address is #prime1#35.

4. Click on Add button to add the breakpoint into the Breakpoints display pane.
5. Click on OK button to exit.

Once executed, Hitool will break the target at the specified address only if and when the expression or count matches.

When PC address is 0x802C, breakpoint will work and emulation will be stopped.

NOTE

More>> button is only for threadx support.

Click on More>> button to display Task Qualify check box and Task list box. You can check Task Qualify and choose the task from the list box in the bottom which lists all the created threads.

Only when the task you choose is running, will the breakpoint you set function only if and when the expression or count matches.

Setting Breakpoint through Breakpoints Window

You can set specific software breakpoint by accessing the Breakpoints Window as follows:

1. Click anywhere in Breakpoints Window, in the pop-up menu choose Add item.
2. In the pop-up Breakpoint dialog box, set software breakpoint.

Setting Conditional Software Breakpoint through Breakpoint dialog box

You can use the Count field to implement a simple conditional breakpoint, the debugger will report halt status only after the breakpoint encountered enough times (specified in the Count field). Also you should use the Break at Location If Condition to implement a

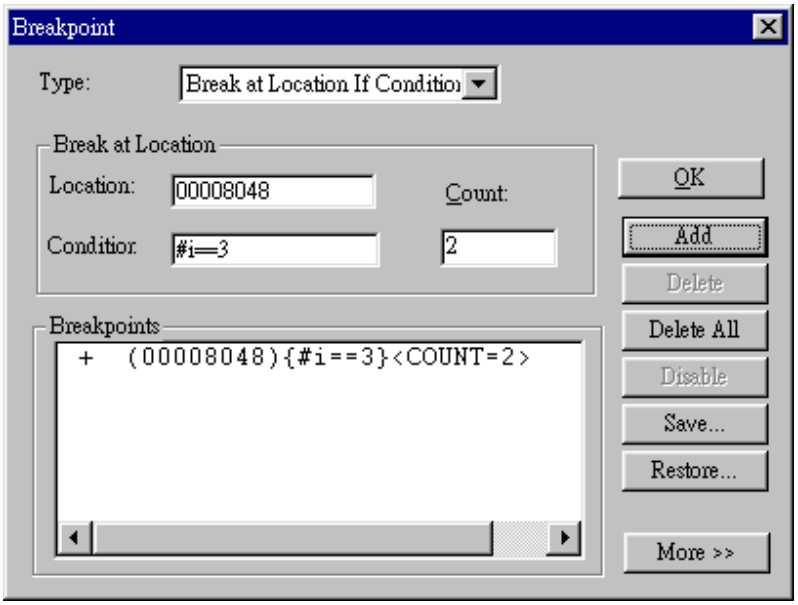
more complex breakpoint. If you want to set more complex breakpoints, you can write a macro with VBScript.

You can set specific conditional software breakpoint by accessing the Breakpoint dialog box as the following:

1. Choose Breakpoint command from the Debug menu.

Then set the conditional breakpoint in the Breakpoint dialog box as illustrated below.

2. Select the breakpoint type " **Break at Location If Condition** " from the **Type** combo box.
3. Fill in the address in the **Location** field, condition in the **Condition** field and Count in the **Count** field.



4. Click on Add button to add the breakpoint into the Breakpoints list pane.
5. Click on OK button to exit.

When meet the condition "#j == 3", count will be increased by one. After count is increased to the count number "2", breakpoint will work and emulation will be stopped.

Breakpoint dialog box

With the Breakpoint dialog box open, you can view and set software breakpoints. The dialog box also allows you to view the status and address of the all the software breakpoints you set.

Opening the Breakpoint dialog box

You can open the Breakpoint dialog box by doing one of the following:

- Locating the Build Toolbar and clicking on the Breakpoint icon.
- Clicking on the Breakpoint menu item under the Debug Menu.

Specifying the Breakpoint State and Type

When a breakpoint is reached, emulation halts before the instruction at the breakpoint address is executed.

Breakpoints can be enabled, disabled, or cleared. An enabled breakpoint is defined and active; emulation breaks when the breakpoint is reached. A disabled breakpoint is defined but is inactive; emulation does not break when the breakpoint is reached.

The two states, enabled and disabled, allow you to turn breakpoints on or off. For example, enable a permanent breakpoint and emulate until that breakpoint is reached. Then, disable the breakpoint and continue emulation. Later, you can enable the breakpoint again and continue emulation.

Deleting Selected Software Breakpoint

In the Breakpoint dialog box, highlight the breakpoint you wish to delete and click on Delete button.

Disabling Selected Software Breakpoint

In the Breakpoint dialog box, highlight the existing enable breakpoint that you wish to disable and click on Disable button. The plus (**+**) symbol prefixing the breakpoint will turn into the minus (–) symbol.

Enabling Selected Software Breakpoint

In the Breakpoint dialog box, highlight the existing disable breakpoint that you wish to enable and click on Enable button. The minus (**–**) symbol prefixing the breakpoint will turn into the plus (+) symbol.

Deleting All the Breakpoint(s)

In the Breakpoint dialog box, click on Delete All button to delete all the software breakpoints you set.

Saving all the Breakpoint(s)

In the Breakpoint dialog box, to save all the software breakpoints you have set, do the following:

Click on Save button. Save Breakpoint dialog box will popup to accommodate saving process.

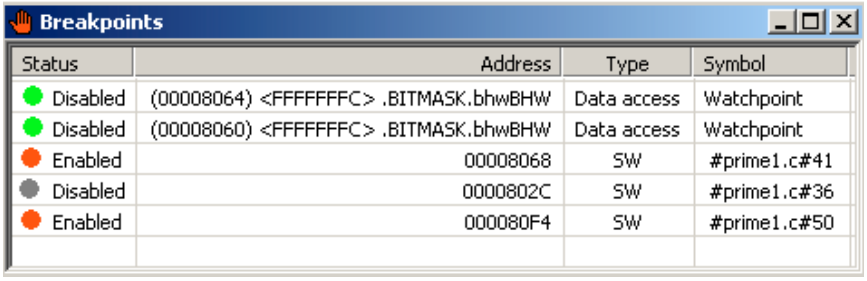
Restoring Saved Breakpoint(s)

In the Breakpoint dialog box, to restore all the software breakpoints you saved, do the following:

Click on Restore button. Choose the directory and filename in the pop-up Restore Breakpoint dialog box.

Breakpoints Window

With the Breakpoints Window open, you can view and set software breakpoints. The Window also allows you to view the status, type and address of all the software breakpoints you set.



Opening the Breakpoints Window

You can open the Breakpoints Window by doing any of the following:

- Clicking on the Debug Window menu item from the View Menu. Then choose Breakpoints item.

Adding Software Breakpoint or Watchpoint

In the Breakpoints Window, highlight the breakpoint and choose Add item from the pop-up menu to invoke the Breakpoint dialog box or highlight the watchpoint and choose Add item from the pop-up menu to invoke the Watchpoint dialog box through which you can add software breakpoint or watchpoint.

Deleting Selected Software Breakpoint or Watchpoint

In the Breakpoints Window, highlight the breakpoint or watchpoint you wish to delete and choose Delete item from the pop-up menu.

Disabling Selected Software Breakpoint or Watchpoint

In the Breakpoints Window, highlight the existing enable breakpoint or watchpoint that you wish to disable and choose Disable item from the pop-up menu. The symbol red dot prefixing the breakpoint will turn into green dot .

Enabling Selected Software Breakpoint or Watchpoint

In the Breakpoints Window, highlight the existing disable breakpoint or watchpoint that you wish to enable and choose Enable item from the pop-up menu. The green dot symbol prefixing the breakpoint will turn into the red dot .

Deleting All the Software Breakpoints or Watchpoints

In the Breakpoints Window, choose Delete All item from the pop-up menu to delete all the breakpoints or watchpoints.

Disabling All the Software Breakpoints or Watchpoints

In the Breakpoints Window, choose Disable All item from the pop-up menu. The all the red dot symbols prefixing the breakpoints will turn into the grey dot symbols.

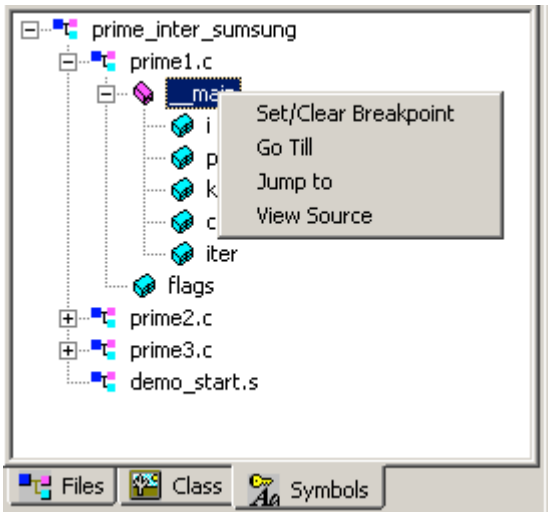
Enabling All the Software Breakpoints or Watchpoints

In the Breakpoints Window, choose Enable All item from the pop-up menu. All the grey dot symbols prefixing the breakpoints will turn into the red dot symbols.

Project Window Symbols Pane

The Project Window Symbols Pane offers the following option for setting software breakpoint within the window.

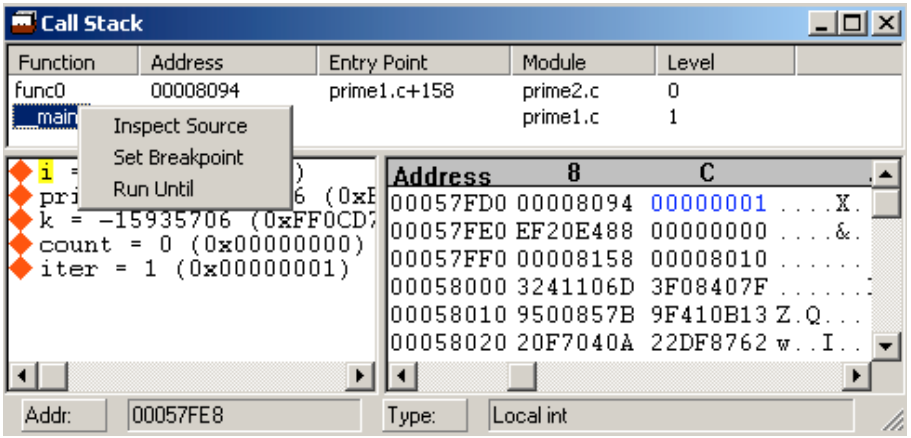
- Breakpoints can also be set by selecting a function in the Symbols Pane of the Project Window. Right-clicking will invoke the pop-up menu and choose Set / Clear Breakpoint item.



Stacks Window

Breakpoints can also be set in the Stack Window when it is active.

- Select a function in the Stack Window and right-click to invoke the pop-up menu.



When the menu appears, select the Set Breakpoint item.

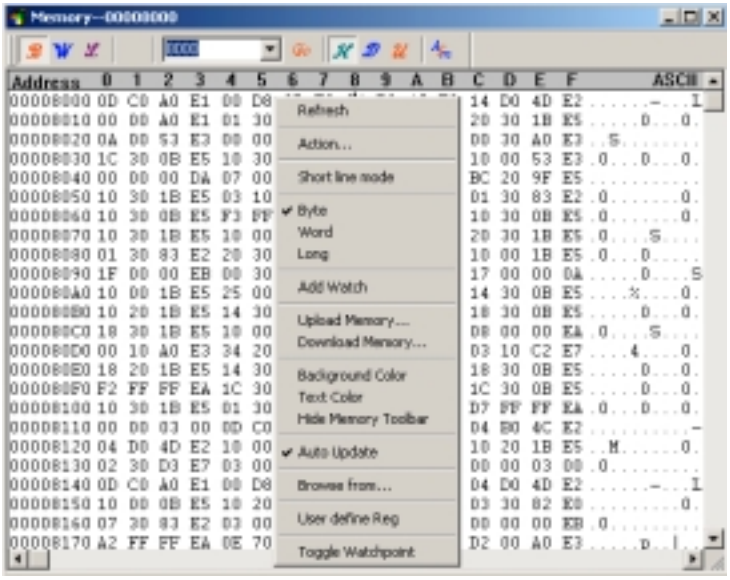
Setting Watchpoints

Setting Watchpoints for Angel series

Memory Window

> ***NOTE***
>
> *For angel series, use the pointed address (4-aligned) to set a bitmask type watchpoint, whose bound is 0xFFFFFFFC, monitoring 8, 16, and 32-bit writes.*
>
> *For wiggler series, use the pointed address to set a data access watchpoint with data mask is 0xFFFFFFFF(data is not cared).*

Point to the content from within the Memory Window then right click on to invoke the pop-up menu as shown below:



Then choose Toggle Watchpoint from the shortcut menu. A watchpoint will be set at the memory address you currently point to.

Setting Watchpoint through Source Window

For Angel series, to set a watchpoint in source window, you can do as following:

1. Active source window.
2. Point to a globale variable, then right click.
3. Choose Toggle Watchpoint item form the local menu.

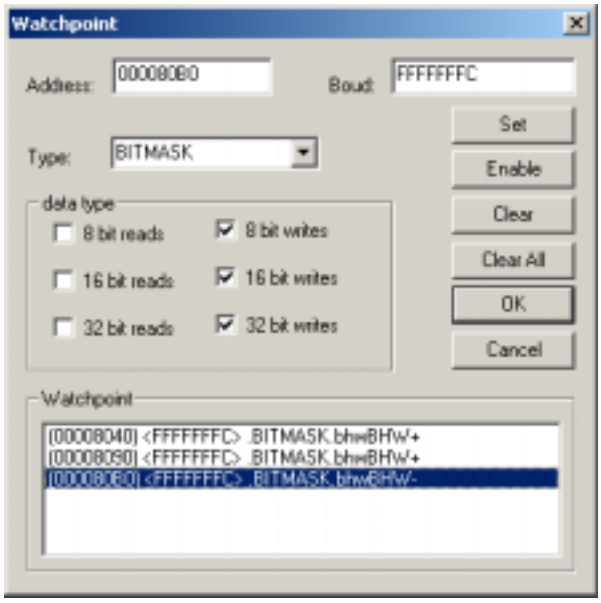
Then a Bitmask type watchpoint will be set at the variable address.

Setting Watchpoint through Watchpoint dialog box

For Angel series, to set watchpoint through Watchpoint dialog box, do as following:

1. Choose Watchpoint command from the Debug menu.
2. Fill in the address in the Address field.
3. Fill the value in the corresponding Boud field.
4. Choose the Watchpoint type from the Type list box.
5. Check the data type option.
6. Click on Set button to set the watchpoint and OK button to close the dialog box.

Thus Watchpoint has been set in Watchpoint dialog box as illustrated below.



Watchpoint Setup with Watchpoint dialog box

Watchpoint dialog box for Angel series

With the watchpoint dialog box open, you can view and set watchpoints. The dialog box also allows you to view the address, option and status of the all watchpoints you set.

Opening the Watchpoint dialog box

You can open the Watchpoint dialog box by doing the following:

- Clicking on the Watchpoint menu item under the Debug Menu.

Deleting All Watchpoints

In the Watchpoint dialog box, click on Clear All button on the dialog box, all the watchpoints you set will be cleared and restore the default settings.

Disabling Watchpoint

In the Watchpoint dialog box, highlight the watchpoint then click on Disable button, the plus symbol + besides the watchpoint will turn into the minus symbol -.

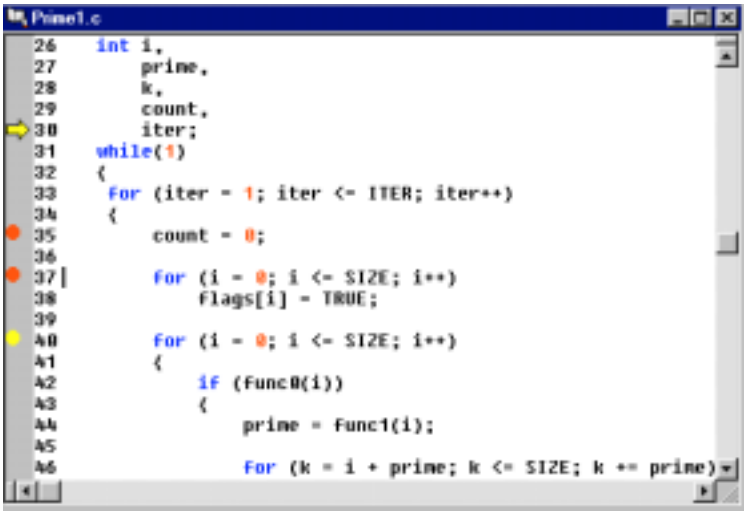
Enabling Watchpoint

In the Watchpoint dialog box, highlight the watchpoint then click on Enable button, the minus symbol - besides the watchpoint will turn into the plus symbol +.

Setting Watchpoints for Wiggler series

Source Window

Click on any line number from within the Source Window to invoke the pop-up menu as shown below:



Options for setting watchpoints include:

- To set a watchpoint you can also simply right click on the left of any source in the Source Window or low-level address in the Mixed Source Window and Disassembly Window.

Bull's-Eye

Point the mouse to the left of a source line, Hitool displays a watchpoint (bull's-eye) cursor. Then click on the source line to set a watchpoint. A watchpoint set symbol (yellow dot) appears to the left margin of the source line.

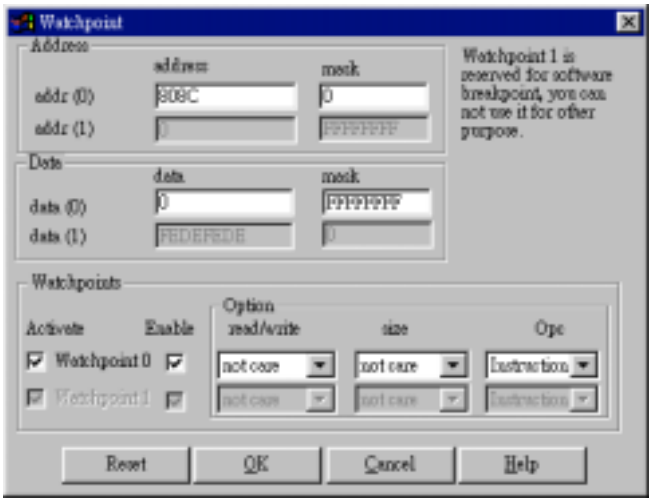
Watchpoint Symbol

Watchpoint dialog box

For Wiggler series, you can set watchpoint in watchpoint dialog box. Do as follows:

1. Choose Watchpoint command from the Debug menu.
2. Check Activate check box beside Watchpoint0 to set Watchpoint0.
3. Fill the value in the corresponding address (0) or data (0) field.
4. Make the further read/write, size. Opc options by select the item from the combo box in Options section.
5. Check Enable to enable the Watchpoint0.
6. Click on OK button to set the watchpoint0 and close the dialog box.

Thus Watchpoint0 has been set in Watchpoint dialog box as illustrated below.



Watchpoint0 Setup with Watchpoint dialog box

Output Window Command Pane

Hitool provides you with powerful commands to directly control software breakpoints and watchpoints instead of operation through window or dialog box. You can use shell command to set watchpoint both for Angel series and for Wiggler series.For further information, see the chapter of “Command Reference”.

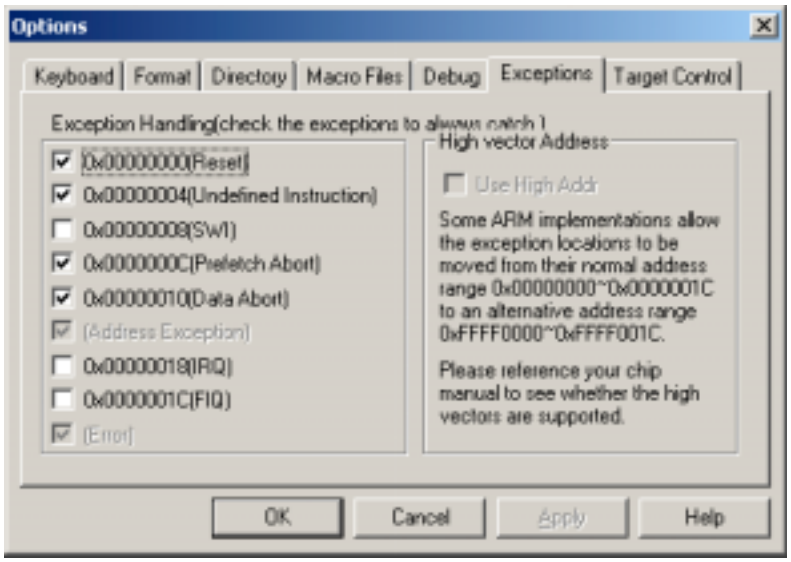
Configuring Exception Trapping Settings

These settings control how ARM and the target handle and share run-time exceptions. Exception Trap Settings may be configured from the Hitool Window GUI.

To configure exception trap settings, click on Options menu item from Options menu. Switch to Exceptions tab, and check the Exception Handling.

For Wiggler series, the debugger sets the corresponding bit of the Vector Catch register to monitor the exception.

But for Angel series, it just sets a DWORD value to Angel debug monitor or Jeeni, which will monitors the exception.



8 Project Window

Project Window Overview

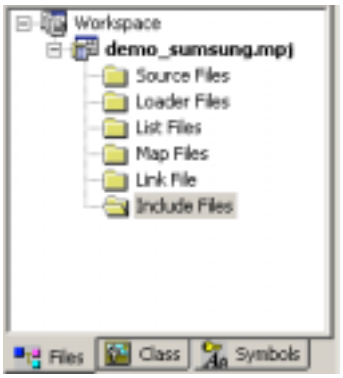
After you launch Hitool, Project window will be open by default.

The Project Window is modeled after the Microsoft Windows Explorer. The window consists of three panes, with the right side Symbols pane providing an outline of the three categories of symbols: Modules, Functions, and Variables. The left side of the window provides an outline view of project categories. And the middle of the window provides all the class information defined in source files.

Working with Project

Files pane makes it easy to configure your embedded software projects and save the project settings into project files, and then restore them with just a few mouse clicks.

Files pane shows the relationships among the files included in the project workspace. The relationships in Files pane are logical ones, not physical ones. It does not reflect the organization of files on your hard disk. Files pane shows the relationships of the source files and its relevant files.



Files pane

With the project open, its name in the workspace is indicated in Files pane in bold. And Hitool will load the code from the executable file automatically if the corresponding executable file exists.

Workspace directory is the root directory of project workspace.

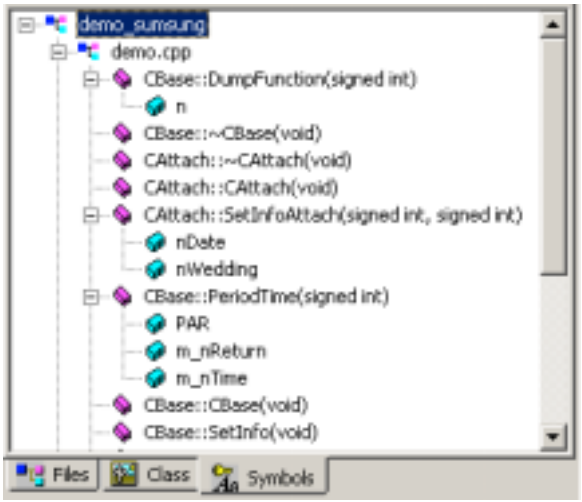
Every project consists of six relative default folders. They are Source Files (which may be separated into three independent folders ARM C Sources, Thumb C Sources and Assembly Sources if the project is for interworking), Loader Files, List Files, Map Files, Link File, Include Files.

You can also create new folder in the project. To do this, highlight the project file then choose Create New Folder from the shortcut menu, in the pop up New folder dialog box fill in the folder name you want to create.

For example, you can create a folder named "Document" and then you can add MS Word document or others into this folder. Then you can view these documents just by double clicking on them. To delete the folder you created or the files you added just press Delete key.

Working with Symbols

The Project Window Files pane displays all the symbols information in source files. You can make full use of the functions with just a few mouse clicks.



Symbols pane

You can expand the tree to view symbol information further. After you expand information thoroughly, you can see all the information of current active project, such as modules, functions and variables listed in the pane.

Point to an item in Symbols Pane and stop for several seconds, all the detailed information such as Address, Attribute, Type and Size of different items will be displayed in a box.

You can debug through Symbols Pane conveniently.

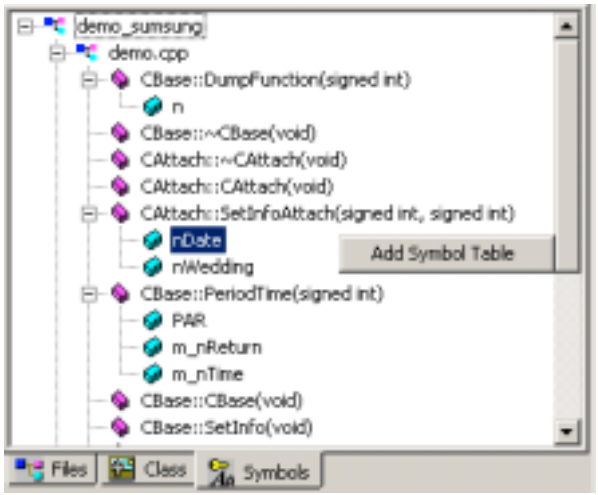
Add or Remove Symbol Table

Hitool supports multi-symbol-table to support Linux programs. After loading the main project, you can add and remove other symbol tables.

NOTE

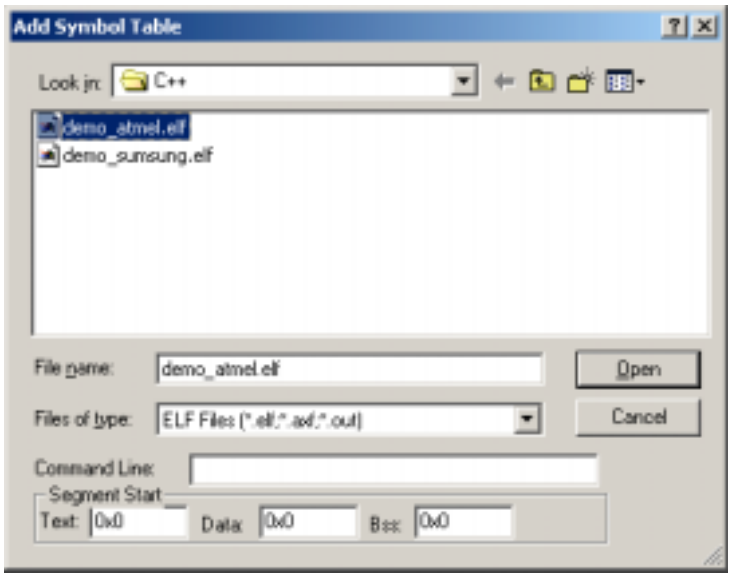
You can not remove the symbol table of the main project.

To do this, simply right click on anywhere in Symbols Pane, choose Add Symbol Table from the shortcut menu.



-or- Choose Add Symbol Table from File menu.

Then in the pop-up Add Symbol Table dialog box shown as below, choose the executable file.



Furthermore, while you add symbol table, you can input offset of your new symbol table. There are three types of offset:

1. Text - text offset affecting the address of code, such as the module and function

2. Data - data offset affecting the address of data, such as global variables

3. Bss - Bss offset

Thus Project nodes are added in symbol list.

NOTE

If you perform reload or normal load , all added symbol tables will be removed.

Drag and Drop Capabilities

The Project Window allows you to read data under any of the following categories: Functions, Modules and Variables.

The Project Window gives you the ability to place items directly into Watch Window.

Modules

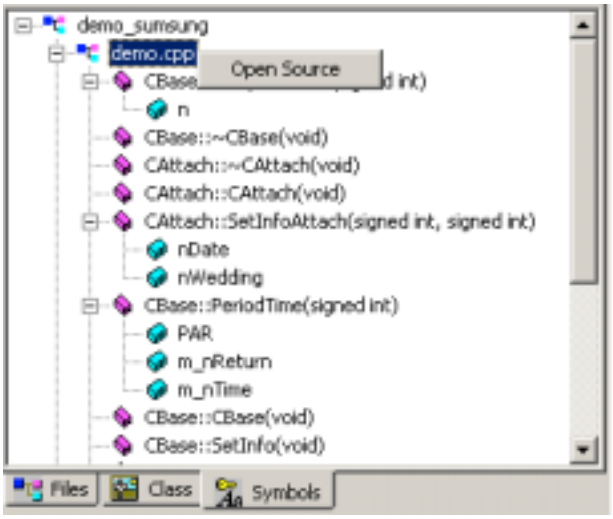
You can expand and contract the modules folder by clicking on the + box to the left of the modules.

By clicking on the module, a listing of the active modules is displayed in Symbols pane, where more detailed information is given.

By selecting a module in the Symbols pane, and clicking again, detailed information about the module is displayed in the right side of the screen.

Shortcut Menu for Modules

Highlight a module then right-click in the window to invoke the following pop-up menu:



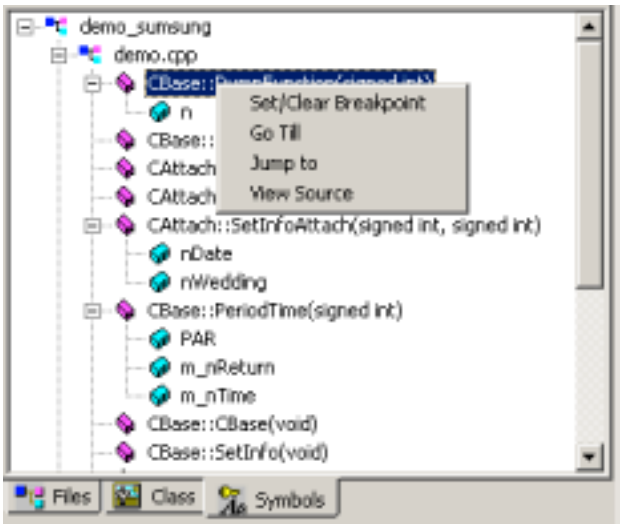
Open Source - After you have selected a module, click on the Open Source menu item to load the module in Hitool's Source Window. Hitool will align the source display and locate to the first executable source line in the procedure specified.

Functions

You can expand and contract the Modules folder by clicking on the + box to the left of Modules. By expanding this view, a listing of functions contained in that Module are displayed and more detailed information is provided.

By expanding this view, you can obtain detailed information on a particular module.

While you still have the Functions folder highlighted, right-click anywhere on the right-hand side to invoke the following pop-up menu:



Set / Clear Breakpoint - After you have selected a function, click on the Set / Clear Breakpoint menu item to set or clear the software breakpoint on the first executable source line in the procedure specified.

Go Till - After you have selected a function, click on the Go Till menu item to emulate to the right start address of that function.

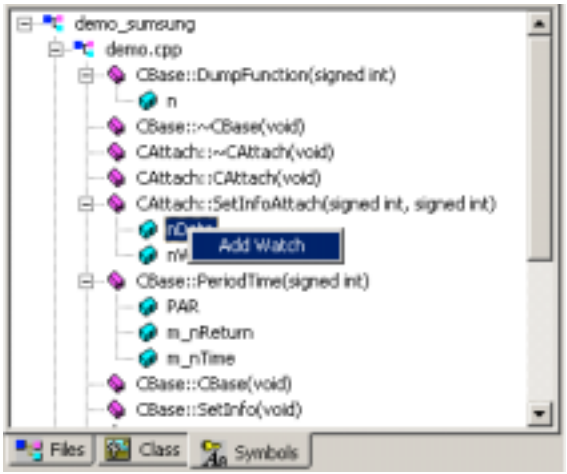
Jump To - After you have selected a function, click on the Jump To menu item to set the target's Program Counter to the start address of the function that you selected.

View Source - After you have selected a function, click on the View Source menu item to load the module containing that function in Hitool's Source Window. Hitool will align the source display and locate to the first executable source line in the procedure specified.

Variables

You can expand and contract the Functions folder by clicking on the + box to the left of Functions. By expanding this view, you can obtain detailed information that displays local variables.

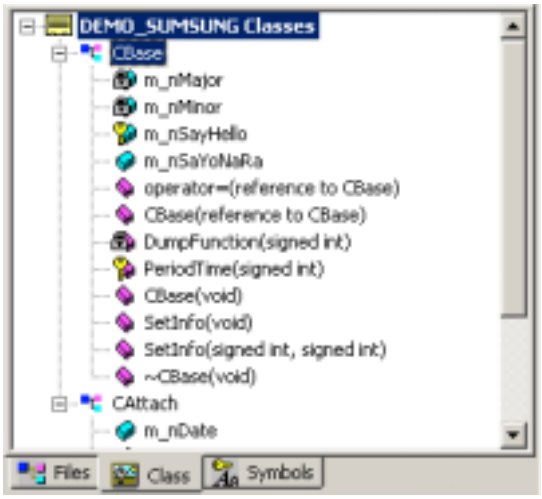
While you still have the Variable highlighted, right-click to invoke the following pop-up menu:



Add Watch - After you have selected a variable, click on the Add Watch menu item to add the variable to the Watch Window's watch list. These variables are automatically updated every time the target stops.

Viewing Class

Class pane shows all the struct and class information defined in source files include public, private and protected members. Every class member includes functions and variables. Point to the item, a tip card will pop up indicating name and type.



Class pane

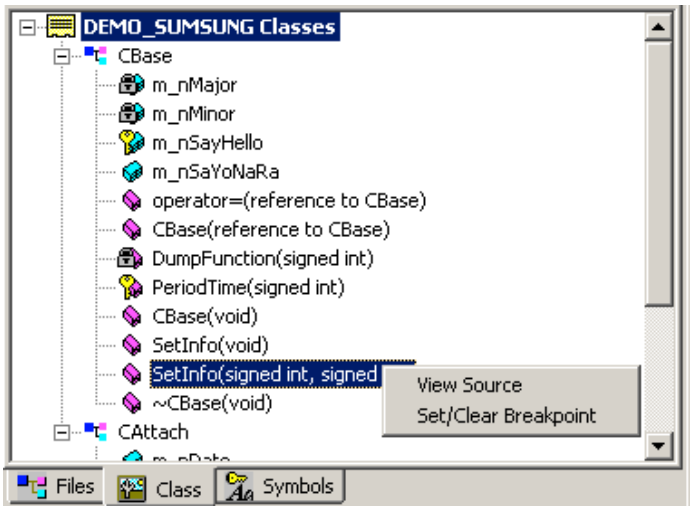
The following table lists the icons in the Class pane and what they indicating.

Icon	Type
	Class
	Struct
	Public Variable
	Private Variable
	Protected Variable
	Public Function
	Private Function
	Protected Function

Functions

You can expand and contract the Class folder by clicking on the + box to the left of Class. By expanding this view, a listing of members contained in that Class are displayed and more detailed information is provided.

While you still have the Functions folder highlighted, right-click anywhere on the right-hand side to invoke the following pop-up menu:



View Source - After you have selected a function, click on the View Source menu item or press Enter key to load the module containing that function in Hitool's Source Window. Hitool will

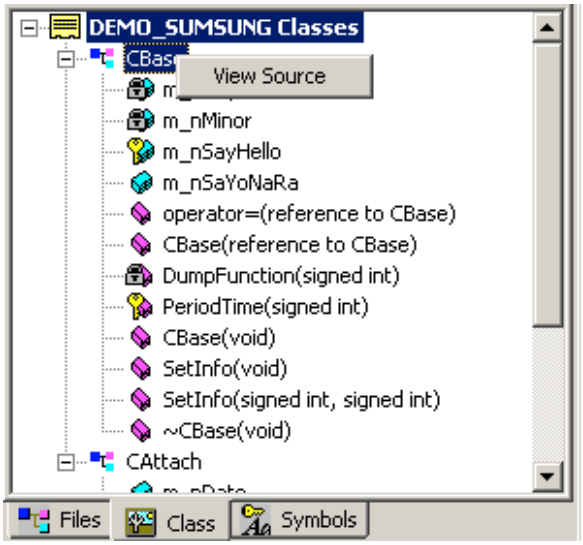
align the source display and locate to the first executable source line in the procedure specified.

Set / Clear Breakpoint - After you have selected a function, click on the Set / Clear Breakpoint menu item to set or clear the software breakpoint on the first executable source line in the procedure specified.

Type Name

You can expand and contract the Type name folder by clicking on the + box to the left of the Type name. By expanding this view, a listing of members contained in that type are displayed and more detailed information is provided.

While you still have the Type name folder highlighted, right-click anywhere on the right-hand side to invoke the following pop-up menu:



View Source - After you have selected a function, click on the View Source menu item to load the module containing that type name in Hitool's Source Window. Hitool will align the source display and locate to the first executable source line in the procedure specified.

Closing the Project Window

To close the Project Window, use the window's control handle (cross) located in the upper right-hand corner of the window.

- Click on the control handle in the upper right-hand corner of the window directly.
- Uncheck the Project window menu item in View menu.

9 Memory Window

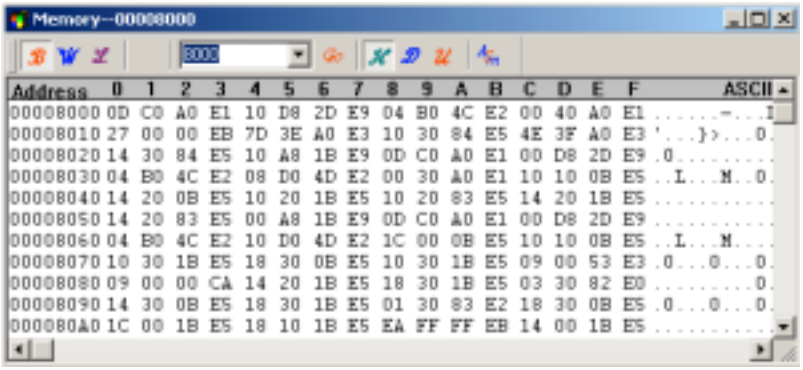
Memory Window

The Memory Window allows you to display and modify target memory.

You can open up to three memory windows where you can view various memory address of the flat memory space. In each memory window, from left to right are:

- The hexadecimal memory addresses.
- The contents of memory, which can be edited. These contents can be displayed with four modes.
- The ASCII equivalent of the memory contents, which can be changed correspondingly with the memory contents.

The ASCII field displays the memory as ASCII characters or a period, a period (.) represents a non-printable character.



Memory Window

Editing Memories

To edit a memory field:

1. Move the cursor into the field you want to change.
2. Type over the field to modify it. A field is defined as either a center column or one of the 16 ASCII characters.
3. Move the cursor out of the field to change the field.

An illegal entry will receive no response from Hitool. Illegal entries include:

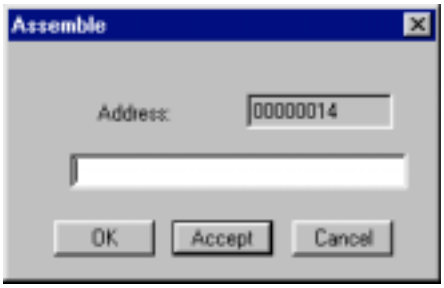
- Non-numeric values in Decimal display.
- Non-hexadecimal values in Hexadecimal display.

The default display for address and data are in Hexadecimal.

Use the Single-line Assembler

To assemble code starting at a specified address, do the following:

1. In the Memory window, click on ASM Code button on the memory toolbar to display disassembly lines of code.
2. Enter exact hexadecimal address and then click on Go button on the memory toolbar.
3. Double click on the disassembly line in the Instruction area to open Assemble dialog box. Fill ARM (or Thumb) instruction in the box. Note that the value in the same line of OP code area is changed accordingly.



-or-

Double click on OP code area to invoke Modify Memory dialog box to enter the value. The content in the same line of Instruction area is changed accordingly.

4. Click on Accept button in the Assemble dialog box

   -or-

   Click on Modify button in the Modify Memory dialog box to write the result into memory and update the Memory window.

5. Click on OK button to quit Assemble dialog box

   -or-

   Modify Memory dialog box.

6. Click on Cancel button to ignore what you have entered and quit Assemble dialog box

   -or-

   Modify Memory dialog box.

NOTE

If you perform some actions to write memory, such as fill, copy, download and so on, you should be within the specific RAM range. For more detailed information, please refer to the related manual.

Memory Action

Choose Memory Action command from Tools menu or Memory command from the shortcut menu to define memory setup. The command displays Memory Action dialog box which contains 5 tabs. You can fully perform memory actions within this group of the dialog boxes.

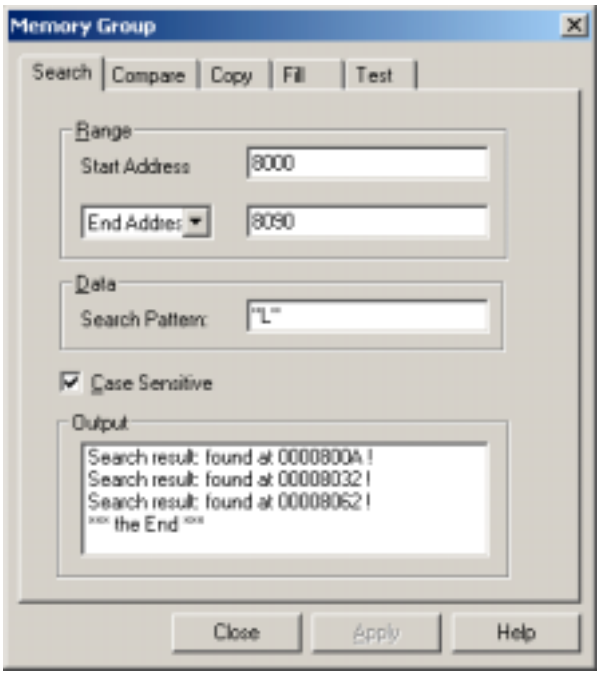
But Hitool only supports Search, Compare, Copy, Fill and Test memory actions. Search, Compare, Copy, Fill and Test actions use the current memory access size setting, which could be set by Access

shell command or through Memory Access Size menu item in Option menu.

NOTE

If you enter one byte data in Modify Memory dialog box, one byte is written; if enter two bytes data, one halfword data is written, if enter three or four byte data, one word data is written.

Memory Search Dialog Box



Where**:**

Start Address Sets the start address of the memory block to perform search.

Sets the end address of the block to perform search in the next field**;**
- End Address sets the end address
- Length sets the address range

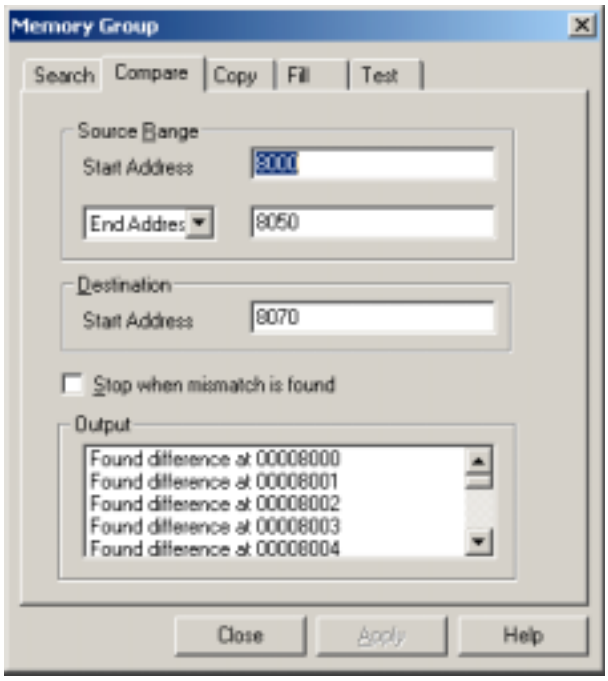
Search Pattern Key in the string of the data pattern to search from the block.

Case Sensitive Check to whether to be case sensitive during search.

Output Message box displays the area where the search data pattern is found. If none is found, “Not Found” will be displayed.

Click on Apply button to start memory search then Close button to quit.

Memory Compare Dialog Box



Where:

Source Range - Start Address — Sets the start address of the first block to compare

Sets the end address of the first block in the next field;
- End Address sets the end address
- Length sets the address range

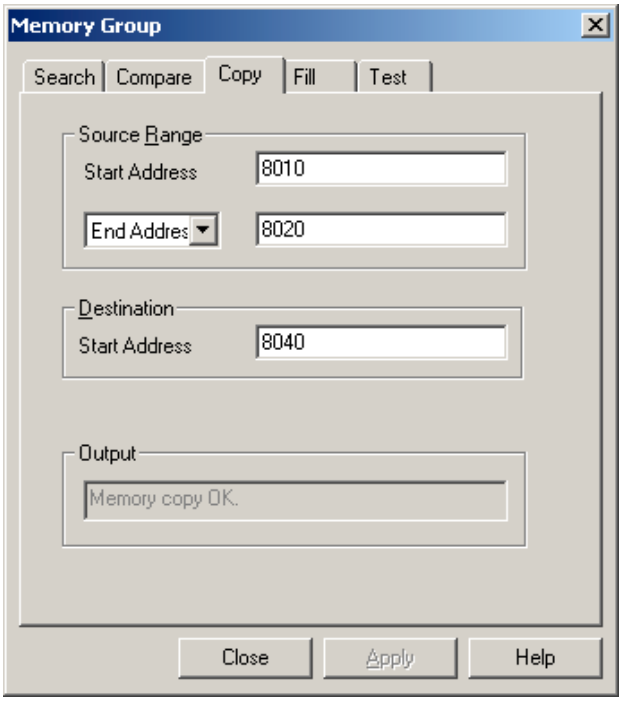
Destination - Start Address — Sets the start address of the second block to compare

Stop when mismatch is found — When selected (checked), the compare command will stop when first mismatch address is found

Output — Message box shows the compare result for the compared block.

Click on Apply button to start memory compare then Close button to quit.

Memory Copy Dialog Box



Where**:**

Source Range - Start Address — Sets the start address of the first block to copy.

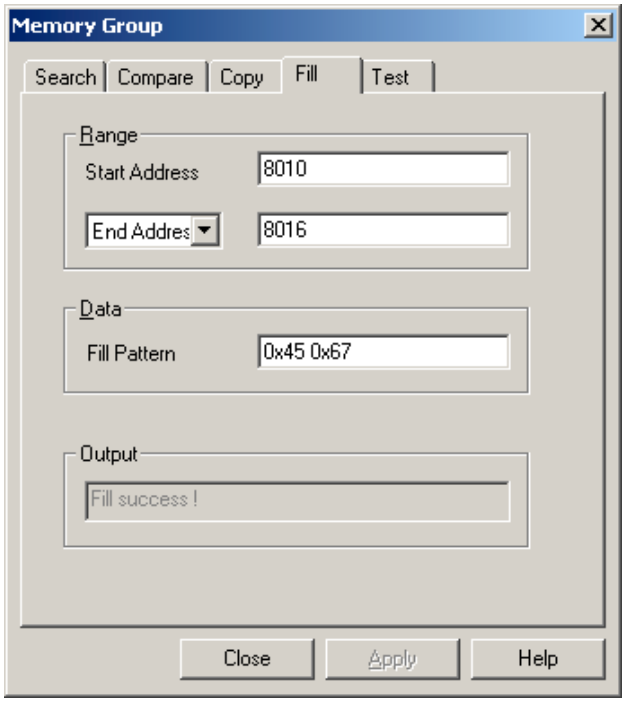
Sets the end address of the first block in the next field**;**
- End Address sets the end address
- Length sets the address range

Destination - Start Address — Sets the start address of the destination block to copy to.

Output — Message box shows the copy result for the copied block.

Click on Apply button to execute memory copy then Close button to quit. You also can see the result directly from the Memory window.

Memory Fill Dialog Box



Where**:**

Start Address Sets the start address of the memory block to be filled.

Sets the end address of the block to be filled in the next field**;**
- End Address sets the end address
- Length sets the address range

Fill Pattern Sets the data pattern of the block to be filled.

Output Message box shows the fill result for the filled block.

Click on Apply button to fill the memory and you will see the result directly from the Memory window. Click on Close button to quit.

Memory Test Dialog Box

Where**:**

Start Address Sets the start address of the memory block to be tested.

Sets the end address of the block to be tested in the next field**;**
- End Address sets the end address
- Length sets the address range

Action Sets the start address of the memory block to be tested.

- Memory Test sets to test memory
- Data Bus Test sets to test the data bus
- Address Bus Test sets to test the address bus

Access Width Sets the width of the memory block to be tested.

- Byte sets the access width as byte
- Half_word sets the access width as half word
- Word sets the access width as word

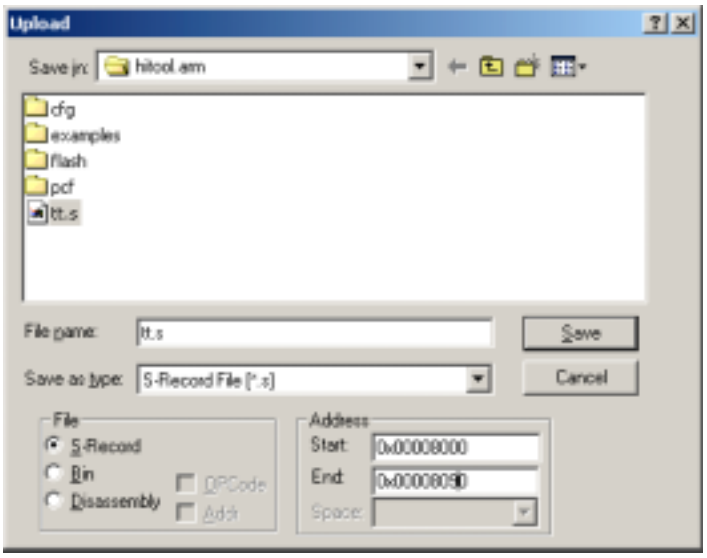
Output Message box shows the test result for the tested block.

Click on Apply button to execute then click on Close button to quit.

Upload/Download Memory

To save the current memory into a file which you can load again next time, do the following:

1. Select Memory Upload from Tools menu of menu bar, then Upload dialog box will popup.
2. In the dialog box, select the path, file format, specify the range of memory then save it into the file you specified.



Upload dialog box

S-Record Choose to upload/download memory in S Record format file.

Bin Choose to dump/download memory in binary format file.

Disassembly Choose to dump/download memory in disassembly format file. It includes **OPCode** and **Addr** two sections.

When dumping, OPCode and Addr indicate whether the OPCode and Address are included in the file.

When Downloading, the OPCode and Addr indicate whether the file includes OPCode and Address.

Start Specify the start address of the memory you want to upload.

--Or-- Specify the start address of the memory you want to upload if the target file does not include address.

End Specify the end address of the memory you want to upload.

Viewing Memory

Hitool offers a number of different options for displaying memory in the Memory Window.

- **Memory Toolbar** – From the Memory toolbar, we can quickly show memory in different modes, navigate through memory or edit memory.

Memory toolbar

Byte	Displays memory elements in byte.
Word	Displays memory elements in half word (two bytes).
Long	Displays memory elements in word (four bytes).
Go	Specifies a numeric to scroll to. On the left there is a combo box for you to input the exact hexadecimal address you want to go to.
Hexadecimal	Displays memory contents in hexadecimal, indicating positive values in all cases.
Decimal	Displays memory contents in decimal, indicating positive values in all cases.
Unsigned Decimal	Displays memory contents in unsigned decimal, indicating positive or negative values.
Asm Code	Displays memory contents in assembly code.

- **Pop-up Menu** – Right-click in the Memory Window to invoke the pop-up menu.

Closing the Memory Window

To close the Memory Window, use the window's control handle located in the upper-left hand corner of the window.

- Click on the control handle and select the Close menu item from the pop-up window control menu.
- Double-click on the window's control handle.

10 ThreadX Window

Hitool ThreadX Window

Hitool works seamlessly with ThreadX, providing detailed debugging for developers as well as full C debugging.

ThreadX examples are included with Hitool software, installed in a directory under ..\Hitool.ARM\ThreadX directory:

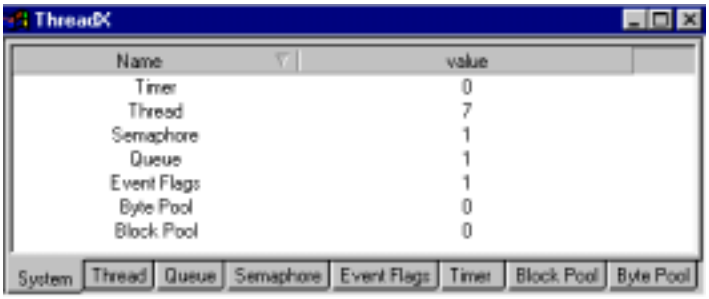
Click on ThreadX item in Tools menu to bring up the ThreadX window.

ThreadX Information window consists of the following eight panes:

- System Pane
- Thread Pane
- Queue Pane
- Timer Pane
- Event Flags Pane
- Semaphore Pane
- Block Pool Pane
- Byte Pool Pane

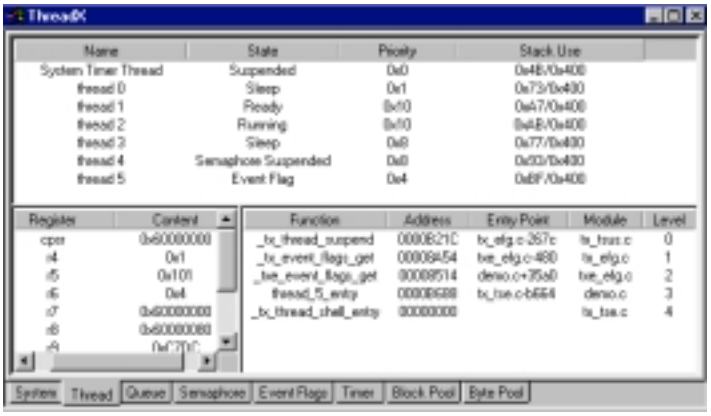
System Pane

System Pane lists seven fundamental components of ThreadX and its corresponding number of current members. Each component is explained below.



Thread Pane

The Thread Pane shows a list of all threads in the system, arranged in the order they were created. When a thread is deleted, it disappears from this list.



The fields displayed for each thread are explained below.

Name - The name of thread.

State – The current execution state of the thread when the system was halted. The thread can be in one of four states:

- **Ready**: The thread is ready to execute. The thread will execute when it is the highest priority thread.
- **Suspended**: The thread cannot run because it is waiting. Threads can wait for time, message queues, event flags, semaphores, and memory, or may be placed in a suspended state on thread creation.
- **Sleep**: The thread sleeps because it is waiting. Threads can be invoked after a time.

- **Queue Suspended**: The thread cannot run because it is suspended by Queue component.
- **Semaphore Suspended**: The thread cannot run because it is suspended by Semaphore component.
- **Event Suspended**: The thread cannot run because it is suspended by Event component.
- **Block Suspended**: The thread cannot run because it is suspended by Block component.
- **Byte Suspended**: The thread cannot run because it is suspended by Byte component.
- **Completed**: The thread has returned from its entry function.
- **Executing**: The thread was executing when the system was halted.

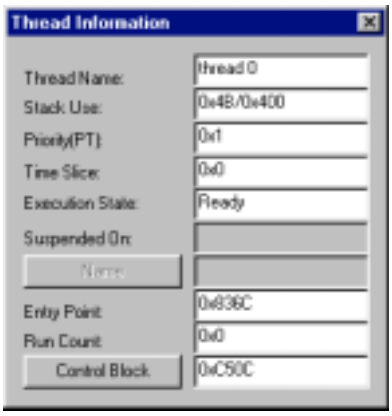
Priority – The priority information of the thread when the system was halted. The first number is the priority level of the thread. The second number, in parentheses, is the preemption threshold of the thread.

Stack Use – The amount of stack currently in use by the thread. Two numbers separated by a forward slash (/) are displayed. The first number indicates the amount of stack the thread has used, and the second number indicates the total amount of stack space allocated to the thread.

Register, Content – The registers list and its value. With emulation halt, when Thread is created and switch between Thread it displays the current registers pop-in stack. If Thread state is running, it will display the registers which are the same as those in Register window.

Function, Address, Entry Point, Module, Level – These columns displays the stack information of the highlighted thread ever called. Otherwise they are blank. They are the same as the contents in the upper pane in Stack Window.

Double-click on, Thread Information dialog will pop-up as the following which shows detailed information about an individual thread.



***Thread** Information dialog*

Time Slice – The timer ticks remaining and the timer ticks given to the thread. There are two numbers separated by a forward slash (/): the first number indicates the remaining timer ticks in this slice, and the second number indicates how many timer ticks the thread will receive when it is subsequently scheduled after it exhausts its current time slice.

Suspended On - The component type on which this thread is suspended. This field shows one of the following values when a thread is suspended: Queue, Semaphore, Event Flags Group, Block Pool, Byte Pool, Sleep, or Suspend Call.

Name - The name of the component on which this thread is suspended, as given when that component was created.

Name Button – Click on to display a view of the component on which this thread is suspended.

Entry Point - The name of the function called upon thread startup. If no debugging information is available for that location, then this field may be displayed as an offset from a known label or an address in hexadecimal format.

Run Count - A count of how many times the thread has been scheduled. When this field is increasing, that indicates the thread is being scheduled and run. A run counter that stays the same may indicate a thread that is unable to run for some reason.

Control Block - Shows the address of the thread control block.

Control Block button - Clicking on to invoke Thread Control Block dialog box which displays the detailed information list of Thread.

Queue Pane

The Queue Pane shows a list of all message queues in the system arranged in the order they were created. When a queue is deleted, it is removed from this list.

The Queue Pane has four columns: Name, Size, Filled and Suspended Threads. Each of these is described below.

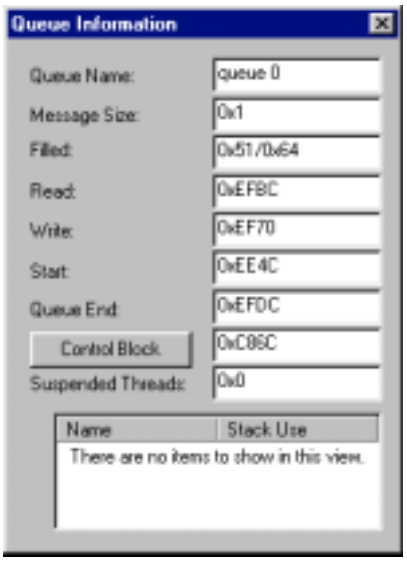
Name – The name of the queue.

Size – The size of each message in the queue. Message sizes range from 1 to 16 32-bit words (ULONGs).

Filled – The value with slash indicating the filled information of message queue. The left section tells the number of message queue currently filled and the right one tells the total size of the queue.

Suspend Threads – The number of threads currently suspended on attempted accesses to this message queue, or 0 if no threads are suspended.

Double-click on, the Queue Information dialog will pop-up as the following and show detailed information about an individual message queue, including a list of threads currently suspended on attempted message queue accesses that have not yet to be satisfied.



***Queue** Information dialog*

Queue Name - The name of the queue.

Message Size - The size of each message in the queue. Message sizes range from 1 to 16 32-bit words (ULONGs). Valid message sizes are 1, 2, 4, 8, and 16 words.

Filled - The number of messages currently stored in the queue awaiting a call. This is shown in the form *enqueued*/*total* where *enqueued* is the number currently stored in the queue and *total* is the total number of messages. Following this, the number of available messages *total* minus *enqueued* is shown for convenience.

Read - The address of the next message.

Write - The address where the next message sent.

Start - The address of the beginning of the message queue storage area.

Queue End - The address of the end of the message queue storage area.

Control Block - Shows the address of the queue control block.

Control Block Button - Click on the button to invoke Queue Control Block dialog box which displays the detailed information list of Queue.

In the bottom, a list of threads currently suspended waiting for the queue. Double-click on any listed thread to display a Thread Information dialog.

Semaphore Pane

The Semaphore Pane shows a list of all semaphores in the system arranged in the order they were created. When a semaphore is deleted, it disappears from this list.

The Semaphore Pane has three columns: Name, Count, and Suspended Threads. Each of these is described below:

Name - The name of the semaphore.

Count - The count of the semaphore. Semaphore counts range from 0 to 0xffffffff.

Suspended Threads- The number of thread currently suspended on an attempt to get this semaphore with a call, or 0 if no threads are suspended.

Double-click on, the Semaphore Information dialog will pop-up as the following and show detailed information about an individual semaphore, including a list of threads currently suspended on attempted semaphore gets that have not yet to be satisfied.

***Semaphore** Information dialog*

Semaphore Name - The name of the semaphore.

Semaphore Count - The count of the semaphore. Semaphore counts range from 0 to 0xffffffff.

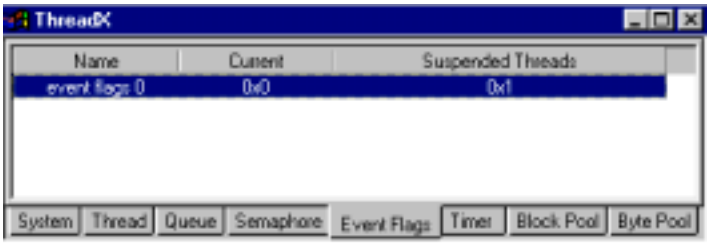
Control Block - Shows the address of the semaphore control block.

Control Block Button - Click on to invoke Semaphore Control Block dialog.

In the bottom, a list of threads currently suspended waiting for the semaphore. Double-click on any listed thread to display a Thread Information dialog.

Event Flags Pane

The Event Flags Pane shows a list of all event flag groups in the system arranged in the order they were created.



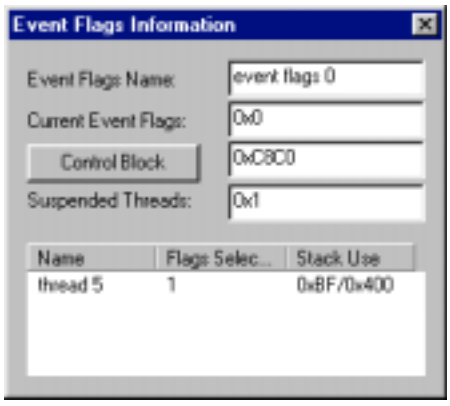
The Event Flags Pane has three columns: Name, Current and Suspended. Each of these is described below:

Name - The name of the event flag group.

Current - The current event flags status in hexadecimal format. Each event flags group contains 32 binary event flags.

Suspended Threads - The number of threads currently suspended waiting for event flags, or 0 if no threads are suspended.

Double click on, The Event Flags Information dialog will pop-up and show detailed information about an individual event flag group, including a list of threads currently suspended waiting for event flags conditions that have not yet to be satisfied.



***Event Flags** Information dialog*

Event Flags Name - The name of the event flag group.

Current Event Flags - The current event flags status in hexadecimal format. Each event flags group contains 32 binary event flags.
Control Block - Shows the address of the event flag group control block.

Control Block Button – Click on to invoke Event Flags Control Block dialog box which displays the detailed information list of Event Flags**.**

Suspended Threads - The number of threads currently suspended waiting for event flags.

In the bottom, a list of threads currently suspended waiting for the event flags. Each column is described in more detail below.

Double-click on any listed thread to display a Thread Information dialog.

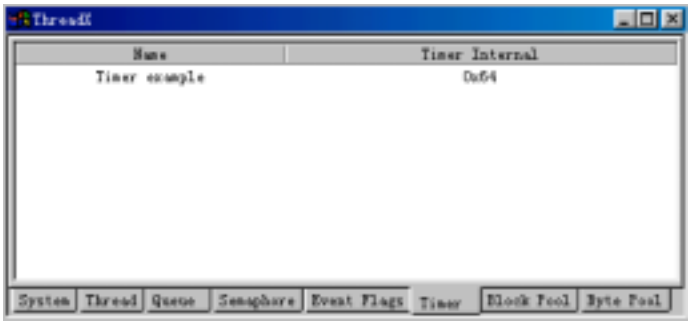
Name - The name of the suspended thread by event flags.

Flags Selected - The event flags that will satisfy the waiting thread's conditions. The first part shows in hexadecimal format the flags that the thread requested. The second part contains one or two characters that show whether the thread is waiting for all or any of the event flags, and whether the event flags will be cleared once the thread's requested event flags are satisfied.

Stack Use - The amount of stack currently in use by the thread.

Timer Pane

The Timer Pane shows a list of all application timers in the system arranged in the order they were created.

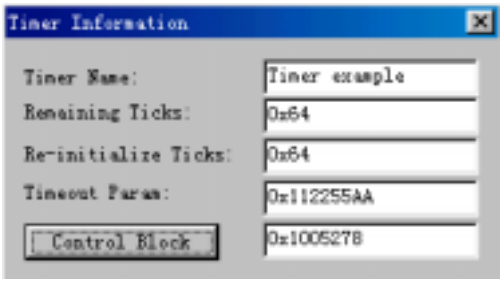


The Timer Pane has two columns: Name, Timer Internal. Each of these is described below.

Name - The name of the timer.

Timer Internal – The combination of some members.

Double click on to invoke the following Timer Information Dialog and show detailed information about an individual timer.



***Timer** Information dialog*

Timer Name - The name of the timer.

Remaining Ticks - The number of initial timer ticks and the timer reschedule tick value. There are two numbers separated by a forward slash (/). The first number specifies the initial number of ticks when the timer is created, and the second number specifies the number of ticks for all timer expirations after the first. A zero in the second number specifies a one-shot timer. Both numbers range from 0 to 0xffffffff.

Re-initialize Ticks - The number of timer ticks when the timer re-initializes.

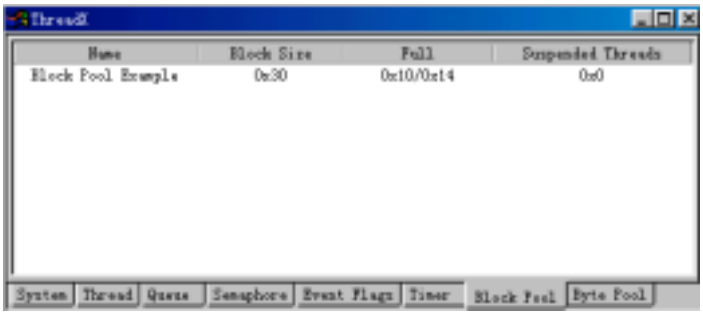
Timerout Param - The parameter passed to the callback function when the timer expires.

Control Block - Shows the address of the timer control block.

Control Block Button – Click on to invoke Timer Control Block dialog box which displays the detailed information list of Timer.

Block Pool Pane

The Block Pool Pane shows a list of all memory block pools in the system arranged in the order they were created.



The Block Pool Pane has four columns: Name, Block Size, Full, and Suspended Threads. Each of these is described below:

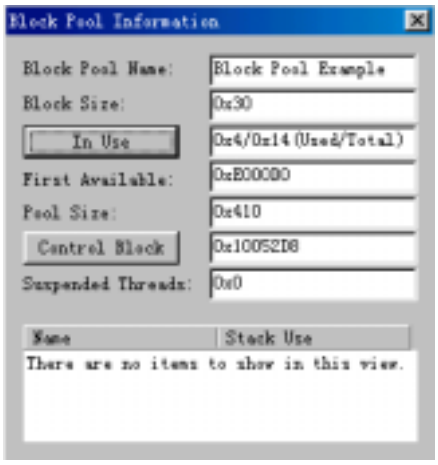
Name - The name of the memory block pool.

Block Size - The size, in bytes, of each memory block in the pool. Block sizes displayed here are rounded up by the **ThreadX** kernel to an even multiple of four bytes to allow suitable alignment for the one pointer of overhead.

Full - The number of memory blocks currently allocated. This is shown in the form (*allocated*/*total*) where *allocated* is the number of blocks currently allocated and *total* is the total number of memory blocks.

Suspended Threads - The number of threads currently suspended on an attempt to allocate a block with a call, or 0 if no threads are suspended.

Double click on, The Block Pool Information dialog will pop-up as the following and show detailed information about an individual memory block pool, including a list of threads currently suspended on attempted block allocations that have not yet to be satisfied.



***Block Pool** Information dialog*

Block Pool Name - The name of the memory block pool.

Block Size - The size, in bytes, of each memory block in the pool. Block sizes displayed here are rounded up by the **ThreadX** kernel to an even multiple of four bytes to allow suitable alignment for the one pointer of overhead.

In Use - The number of memory blocks currently allocated. This is shown in the form (*allocated*/*total*) where *allocated* is the number of blocks currently allocated and *total* is the total number of memory blocks. Following this, the number of available blocks (*total* minus *allocated*) is shown for convenience.

In Use Button - Displays a Block Pool Contents dialog showing specifically which blocks are allocated.

First Available - A pointer to the first available memory block, or zero if the block pool is completely allocated. The address displayed is actually four bytes before the memory block. These four bytes of overhead contain a pointer. Available memory blocks are kept in a singly linked list starting with the first available block. In allocated blocks, this pointer points to the memory block pool control block which allows blocks to be released on a call without specifying the block pool from which the block was allocated.

Pool Size - The number of bytes in the memory block pool storage area.

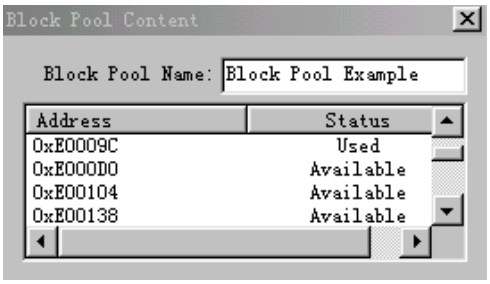
Control Block - The address of the memory block pool control block.

Control Block Button - Click on to invoke Block Pool Control Block dialog box which displays the detailed information list of Block Pool.

Suspended Threads - The number of threads currently suspended on an attempt to allocate a block from this memory pool with a call.

A list of threads currently suspended on an attempt to allocate from this memory pool. Double-click on any listed thread to display a Thread Information dialog.

The Block Pool Contents dialog shows a list of all blocks in a memory block pool.



***Block Pool** Content dialog*

Double-click on any listed memory block to view the contents of memory at that location.

In the button is a list of memory blocks in this block pool. Each column is described in more detail below. Double clicking on any listed memory block will view the contents of memory at that location.

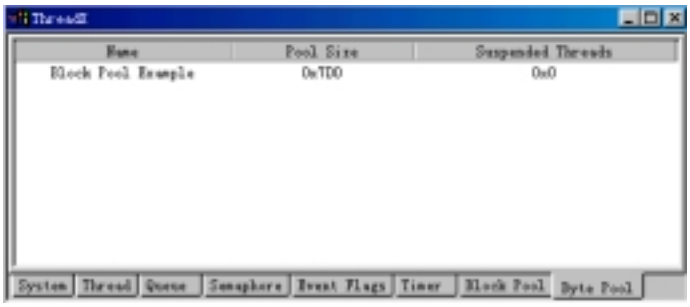
Block Pool Name - The name of the memory block pool.

Address - The address in memory where the block resides. The four bytes immediately previous to this address contain a pointer value. If that pointer points to the memory block pool control block, the block is in use. Otherwise, it is available.

Status - Either In Use or Available depending on the value of the pointer preceding the memory block, as described in Address, above.

Byte Pool Pane

The Byte Pool Pane shows a list of all memory byte pools in the system arranged in the order they were created.



The Byte Pool Pane has four columns: Name, Pool Size and Suspended Threads. Each of these is described below:

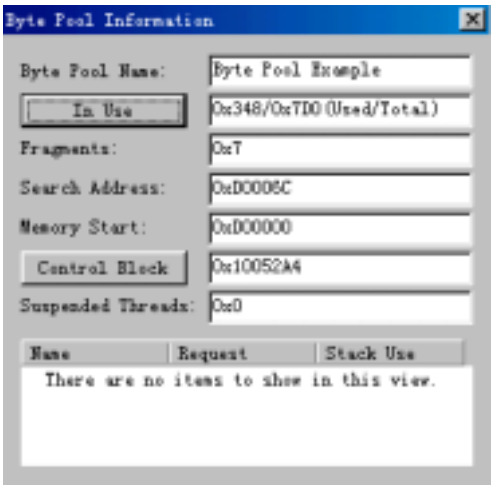
Name - The name of the memory byte pool.

Pool Size - The size of memory pool.

Suspended Threads - The number of threads currently suspended on an attempt to allocate memory from the pool with a call, or 0 if no threads are suspended.

Double click on, the Byte Pool Information dialog will pop-up as the following and show detailed information about an individual memory byte pool, including a list of threads currently suspended on attempted memory allocations that have not yet to be satisfied.

If a memory byte pool is deleted while a Byte Pool Information dialog for that pool exists, the Byte Pool Information dialog will not automatically disappear. It will continue to show the contents of the memory byte pool control block.



***Byte Pool** Information dialog*

Byte Pool Name - The name of the memory byte pool.

In Use - The number of bytes currently allocated from this pool. This is shown in the form *allocated*/*total* where *allocated* is the number of bytes currently allocated and *total* is the total number of bytes. Following this, the number of available bytes *total* minus *allocated* is shown. Note that the available byte count does not compensate for the two pointers of overhead that each memory fragment requires.

In Use Button – Click on to display a Byte Pool Contents dialog showing all allocated and unallocated fragments in the byte pool.

Fragments - The number of fragments in the memory byte pool.

Search Address - A pointer to the first unallocated memory fragment. This is set to the last fragment that was released.

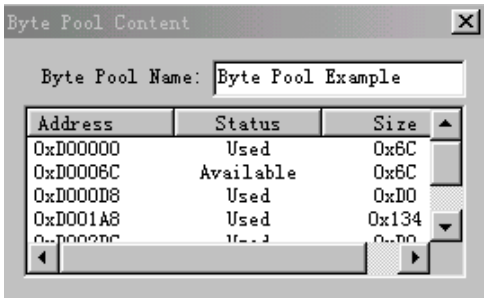
Memory Start - A pointer to the start of the byte pool.

Control Block - The address of the memory byte pool control block.

Control Block Button – Click on to invoke Byte Pool Control Block dialog box which displays the detailed information list of Byte Pool.

Suspended Threads - The number of threads currently suspended on an attempt to allocate a byte from this memory pool with a call.

A list of threads currently suspended on an attempt to allocate from this memory pool. Double-click on any listed thread to display a Thread Information dialog.



***Byte Pool** Content dialog*

The Byte Pool Contents dialog shows a list of all fragments in a memory byte pool.

In the button is a list of fragments in this byte pool. Each column is described in more detail below.

Byte Pool Name - The name of the memory byte pool.

Address - The address in memory of the fragment. The four bytes immediately previous to this address contain a pointer value. If that pointer points to the memory byte pool control block, the fragment is in use. Otherwise, it is available.

Status - Either In Use or Available depending on the value of the pointer preceding the fragment, as described in Address, above.

Size - The size of the byte pool fragment in bytes. All byte pools end with a zero-byte fragment that is shown as 0 (End).

11 Miscellaneous Tools

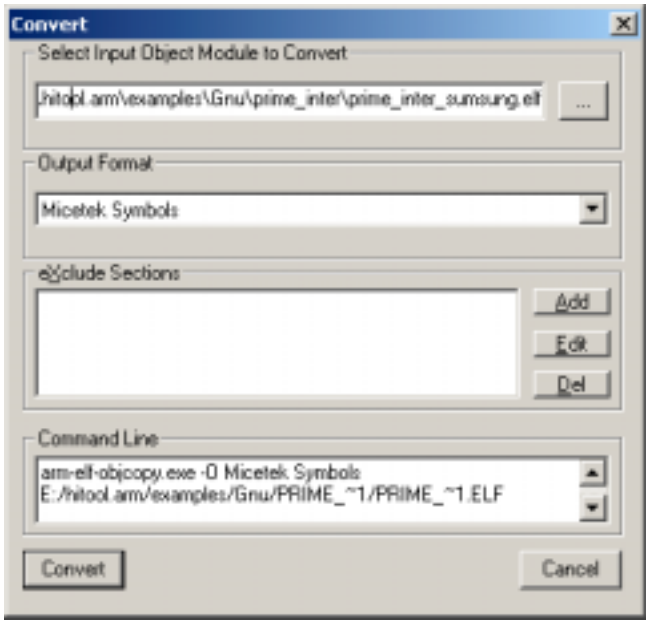
Convert

Hitool offers feature of converting the code part of a target .elf file into binary, hex, s-record format file or the symbol part of a target .elf file into symbol table file (.st) and hash table file (.ht) so as to program Flash or load and it is available only after you open a project.

It needs dos application ARM-eabi-objcopy support. If there is no application, a message will appear in the status pane.

To convert, do the following:

1. Choose Convert menu item from Tools menu. A Convert dialog box will popup up as below.



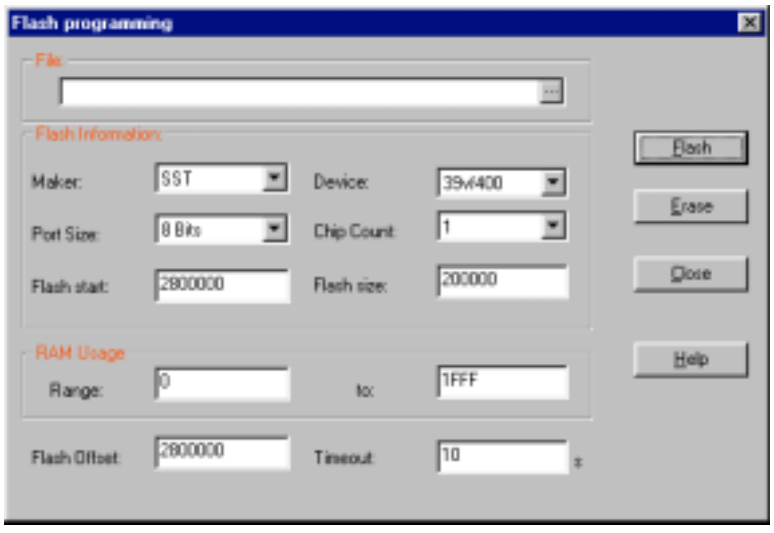
2. Choose the executable file.
3. Set the Output Format from the list box.
4. Click on Add button or Del button to add or delete the sections. The default sections for your choice are .text, .data, .bss, and .rodata. All the sections you want to remove during convert will be listed in eXclude Sections. At the same time the equivalent Command Line text box is displayed at the bottom of Convert dialog box.
5. Click on Convert button to execute converting.

New format file whose name is the same as the executable file name will generate in the current directory and you can use it to test memory flash or load.

Programming Flash with the ARM Utilities Panel GUI

Hitool offers access to the complete Flash programming features of ARM.

By default, the dialog will be set-up to use the newer and faster TF (target Flash) programming method.



Maker Selects the Maker from the list box

Device Selects the corresponding device of the selected maker from the list box

Port Size Selects the port size 8 or 16 Bits from the list box

Chip Count Selects the count of flash chip. The value for your

	selection will vary according to the port size you selected before.
Flash start	Sets the flash start address
Flash Size	During flash, sets the total flash memory size. During erase, sets the size of flash you want to erase. It is no more than the total chip size.
RAM Range	Sets the start RAM address
To	Sets the end RAM address
Flash Offset	Sets the start address you want to erase or flash
Timeout	Sets the timeout to flash or erase

Programming Flash - Step by Step

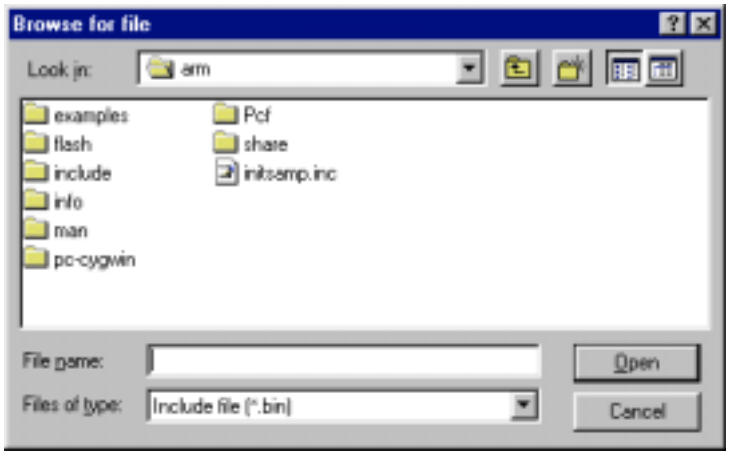
This section provides you with a step-by-step approach to using the Flash programming GUI in the Flash programming dialog box.

Step 1: Invoke the Flash programming Dialog Box

Open the Flash Programming dialog box by selecting the Flash Programming item on the Tools menu item.

Step 2: Select a File to Program Into Flash

In the Flash programming dialog box File field, enter a filename or click on the button to choose a file in the following Browse for file dialog box. In general, the Flash GUI always remembers the last file that you have chosen to Flash.



Click on the file you want to program into Flash, and then click on the Open button to close the dialog and bring the chosen file back to the main Flash programming dialog.

NOTE

All files for Flash programming MUST be in the .BIN format.

Step 3: Enter the Flash Information Address of Your Target

Choose Maker, Device and Port Size from the list box.

Flash (and Optionally Specify Custom Erase) need to know where your target Flash is located on your board. You have two options here:

1. Enter the starting address and size of your device. In this scenario, Hitool will calculate the ending address of your device(s), and erase the specified Flash when you tell it to do so.

2. Optionally, you may enter only the address range of your Flash device(s) for which you want to erase. You would use this feature when you only want to erase selected sectors of Flash or perhaps selected Flash devices in a bank of Flash.

Step 4: Enter a Target Workspace Address

The high speed programming method requires a small amount of Target RAM to perform the Flash algorithm. You need to tell Hitool where write-able RAM is located on your target for this purpose. So fill in starting address and end address in RAM Range field and to field.

Step 5: Begin Erasing and/or Programming Flash

With the Flash programming feature properly configured, you can now choose to Erase or Flash your target Flash.

Logging and Running Include

Hitool can record the shell commands and the result that are sent down to the target.

The log file can be saved in text format but which can use any three characters as file extension. File name can be appended to contain further commands.

Logging On/Off

To record a command file, with or without the corresponding result, specify a valid file name and path in the Command pane first. The Log On/Off toggle shortcut menu is used to begin and end your recordings and will always display the appropriate status in the Command pane.

Writing Include

If you have previously recorded shell commands, you may edit it manually.

The following table lists all the sub-commands used when you write include file:

Sub-commands	Description
****	Used before stop execution.
{	Start sign used at the beginning of a group of sentence.
}	End sign used at the end of a group of sentence.
:label	Used to set the location label where to go to.
BELL[count]	Used to set the time of beep when executing batch file.
ELSE	Conditional branch sentence used together with **IF** or **IF_NOT** sentence.
ERROR {ABPRT|CONTINUE}	Used to set stopping or continuing execution when an error happens.
GOTO label	Used to jump to the label location and start execution.
ICE Command	Used to set to use the shell commands Hitool supports.
IF "condition"	Used to set when the condition in quotation marks is matched, perform the sentence in the braces.
If (C-Expression)	Used to set when the C-Expression in parentheses is matched, perform the sentence in the braces.
IF_NOT"condition"	Used to set when the condition in quotation marks

	is mismatched, perform the sentence in the braces.
IF_NOT (C-Expression)	Used to set when the C-Expression in parentheses is mismatched, perform the sentence in the braces.
LOOP	Used to start loop.
LOOPEND [count]	Used to end loop after specified time.

Running Include

If you have finished writing include, you may invoke Run Include dialog box by clicking on the Run Include item from the shortcut menu of Command pane. Select an Include file then click on the Open button to run include file. All recorded commands are actually re-sent to the emulator and target during running and the commands and results are displayed in the Command pane as they are executed.

Writing and Using Macro

To perform some functions automatically, Hitool provides Macros function with VBScript. Some examples have been provided in the Macro directory of Hitool setup directory. You can load it directly.

To load a macro, do the following:

1. Select Option command from Option menu. In the popup Options dialog box switch to Macro Files tab.
2. Click on Browse button then select the Macro file in the dialog box.
3. Set a checkmark right to the path of macro file then click on OK button to load it.

If some errors exist in the macro file, Hitool will popup an error message dialog box. If no error message dialog boxes popup, that means the macro file is loaded successfully.

You can check load on startup option to load the macro when startup Hitool.

To remove loaded macro file, simply remove the checkmark beside the path of macro file in Macro Files tab.

After you loaded macro file, you can choose Customize from the Option menu to open Customize dialog box. In the Customize dialog box Commands tab, you choose macro category to view the functions defined in macro file.

Hitool enables you to write macro using the objects introduced in Hitool On-line help with VBScript language frame with any edit tools such as Notepad. For detailed information about writing macro with VBScript, please refer to "Advanced Topic" in Hitool On-line help.

NOTE

Before you end the macro, be sure to use a pair of single quotes without any characters or space between them or behind them.

To edit macro, do the following:

Choose Edit Macro command from Tools menu or click on Edit button in Macro Files of Options dialog box , then Macro dialog box will popup. In the dialog box, you can Run or Edit Macro.

Customizable Toolbars

Hitool provides a customizable toolbar for executing common actions using *only* a mouse. Tasks such as stepping, issuing a GO or Reset command can be accomplished with a just a simple click of the mouse. With the Hitool toolbar you can also:

- Move any one of the toolbars to another location on the screen.
- Hide a toolbar or entire group of toolbars.
- Modify an existing toolbar to better suit the needs of your project.
- Add or rename a custom toolbar.
- Use Hitool’s predefined set of commands to create toolbar.

Moving Toolbars

This function allows the user to drag each of the toolbars. The toolbars can then be docked anywhere on the screen, including the sides and bottom.

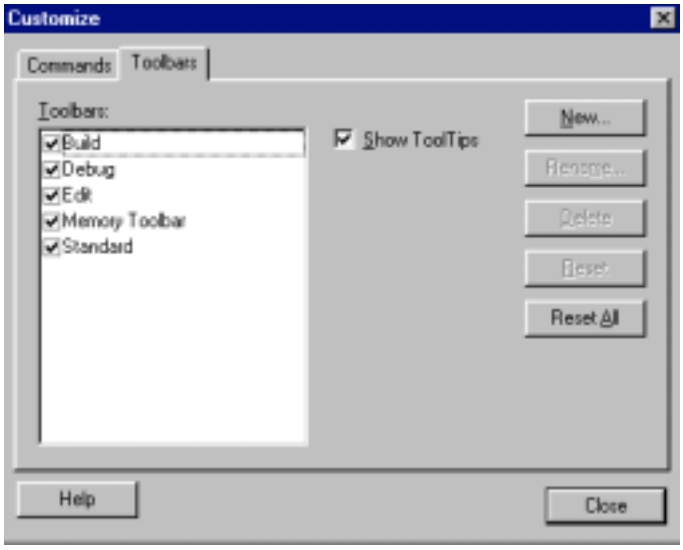
To access Hitool's full range of tool functions, click on the Option menu and select the Customize item from the drop down menu, allowing quick and easy access to its many functions.

Customizing Toolbars

Choose Customize item from the Option menu. Customize dialog box open and switch to Toolbars tab.

With this open, you have the ability of placing toolbars into other locations, including any toolbars that you may have created. By clicking and dragging the toolbar, you can place it into the right location, allowing for greater efficiency.

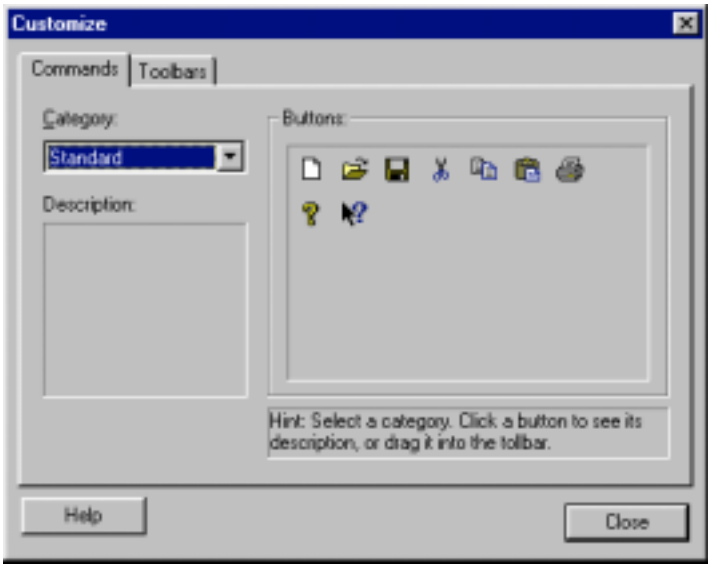
Toolbar Tabs



- Allows you to hide or display an entire toolbars by checking the box to the left of the toolbar name. When the check can be seen, the toolbar is visible, as when the check is gone, the toolbar is hidden.
- Add a custom toolbar.
- Re-name a custom toolbar. But you can not rename the default toolbars.
- Delete custom toolbar.
- Restore the default toolbar settings.

Commands Tab

With access to Hitool's set of predefined commands, the user has the ability, with the *Commands* tab open, to place these into toolbars. Simply click on the icon and drag to place the icon into the desired location or toolbar.



- Allows you to see each category and corresponding commands.
- Gives a set of predefined commands that can be used in a modified toolbar.

Action Record

Hitool provides the function permitting you to record, playback, save all of the screen actions you performed during debugging. When you select the Action Record menu item from the Tools Menu, you invoke the Action Record toolbar button. This window contains six command buttons as the following:

Recording

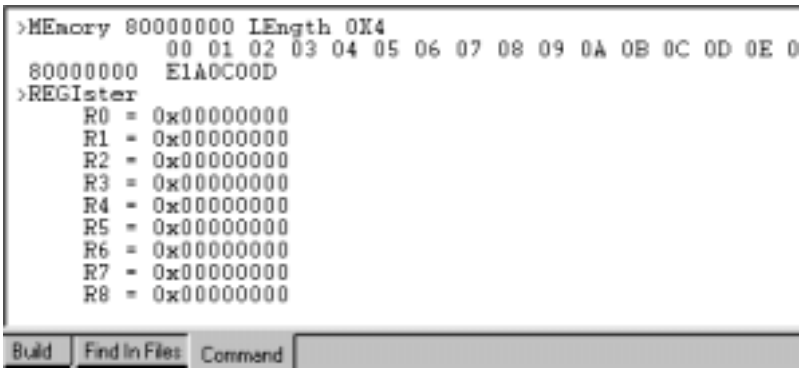
To record the screen actions you performed, just click on Start button. Hitool will record the actions in the file.

Playback

If you have previously recorded screen actions, you may click on Playback command button to replay the screen actions you performed.

12 Command Reference

Shell Command



In the Command Pane, you can key in shell commands instead of window menu commands and see the results.

Hitool features an auto-expansion function which allows you to input command names with a shorthand or abbreviated entry. This shorthand entry is shown in upper case in each command name. The command window status bar is also sensitive. It displays the full syntax and parameters of the command of what you are entering.

Refer to Shell Commands in on-line help for detailed description of each command. You can enter shell commands in the Command pane of Output window instead of window menu commands.

Syntax Notational Conventions

Notation	Meaning
COMMANDNAME commandname CommandName	Command names and aliases may be entered in upper- or lowercase.
[option]	Brackets indicate that the item can be repeated zero or 1 time. Brackets are used to define syntax and should not be entered, unless otherwise noted.
option_1 | option_2	A vertical line separates options, one of which can be selected.
{option_1 | option_2}	One of the options must be selected. Do not enter the braces.
"string_constant"	String constants must be surrounded by double quotation marks.
address	A hexadecimal data or symbol representing an address.

Commands and System Variables Grouped by Function

To Do	See
Assembly/Disassembly	Assemble Command
	Disassemble Command
Breakpoints	Breakpoint Command
	CLear Command
Watchpoints	Watchpoint Command
Emulation	Go Command
	HAlt Command
	Jump Command
	Step Command
	RESTart Command
	RESet Command
Help	Help Command
Load Code	LOad Command
	Upload Command
Memory	ACcess Command
	BYte Command

	COMpare Command
	Copy Command
	Fill Command
	Memory Command
	Test Command
	LONg Command
	Word Command
Register	Register Command
Shell	ALias Command
	Include Command
	LOG Command
Symbols	SYmbol Command
	SSave Command
	SRecall Command
	WAtch Command
	QUEry Command

Here we list all the shell commands Hitool supports in details in alphabetical order.

$variable

$variable [=value]

=value	Defines an integer value or C expression to a variable.

Keyboard Examples:

$a=1	Define variable "a" as the integer value 1.

ACcess

ACcess [-B | -Hw | -W]

ACcess	specifies the memory access format (Word, Byte, Half Ward).
-B	specifies access the memory in Byte format.
-Hw	specifies access the memory in Half Word format.
-W	specifies access the memory in Word format.

This command is to set the format to access the memory.

Keyboard Examples:

ACcess -B	Specify the type of access the memory into Byte format.

ALias

ALias [original_name new_name]

original_name	specifies the system defaultly used Shell command name.
new_name	specifies the new name for the system Shell command.

This command is to rename or shorten the system Shell command. One cannot use it to a Shell command which has been already renamed.

Keyboard Examples:

Alias Memory MM	Rename the Memory command as MM.
MM 0 FF	Use the alias shell command.

Assemble

Assemble [adr]

adr specifies the memory location where Hitool is to begin storing the assembly language program input by the user. It may be a symbol name, line number or an unsigned value.

Show the assembly equivalent of the entered string and write the instruction bytes to memory at the last known address. If memory location is not specified, assembly begins from current program counter (PC). Press “Escape” on the keyboard to stop inputting the instruction.

This command only accepts the instruction set controlled by the current CPU state. For example, for ARM/Thumb state, it will accept CPSR T bit instruction.

NOTE

Assemble command only supports Hexadecimal, and the operators are divided by space or comma.

Keyboard Examples:

1. Assemble 80000010 Input assembly instruction from program address 80000010
2. Assemble Input assembly instruction from current PC
3. Create #ww 80000060

   Assemble #ww
4. Assemble #prime1#32 Input assembly instruction from address of line 32 in prime1 source file

Breakpoint

BReakpoint [adr]

adr specifies the memory location of hardware execution breakpoint. It may be a symbol name, line number or an unsigned value.

If no parameters are specified, all the breakpoints will display.

To delete breakpoints, refer to Clear command.

Keyboard Examples:

BReakpoint 80000060 Set breakpoint at address 80000060 in the memory.

BYte

BYte adr datum1 [..datum32]

adr Specifies the location where the Memory modify operation is to begin. It is a hexadecimal address representing a hex address in the memory space of the emulation CPU. It may be a symbol name, line number or an unsigned value.

Datum1.. datum32 is a hexadecimal value up to 1 bytes(0~FFH).

Keyboard Examples:

BYte 400 1 2 3 Write 1, 2, 3 data to memory 400 to 402.

CLear

CLear {All | adr | HW0 | HW1}

ALl specifies to clear all execution breakpoints and Watchpoints.

adr specifies to clear a particular breakpoint.

HW0 specifies to clear Watchpoint 0.

HW1 specifies to clear Watchpoint 1.

This command is ineffective when emulation processor is running. To add breakpoints, refer to Breakpoint command. To add watchpoints, refer to Watchpoint command.

Keyboard Examples:

CLear ALl Clear all SW breakpoint and Watchpoint settings.

COMpare

COMpare adr1 {adr2 | Length length} adr3 [-B | -Hw | -W]

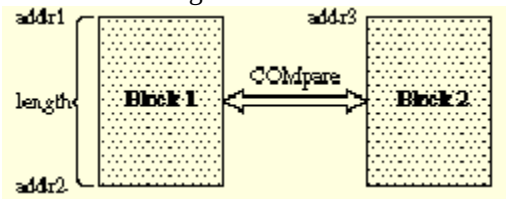
adr1 specifies the starting address of Block 1, the first of the two memory ranges to be compared. It is a hexadecimal address in the memory space of the emulation CPU.

adr2 specifying the end address of Block 1. It is a hexadecimal address in the memory space of the emulation CPU.

Length is the required prefix when specifying a "length" value.

length is a decimal value specifying the memory size of Block 1 (0x1 -0x100000000).

adr3 is a hexadecimal address in the memory space if the emulation processor specifying the starting address of Block 2.



Referring to the diagram above. Block 1 is compared with Block 2. If the blocks match exactly, Hitool will display:

"No different data"

If the blocks do not precisely match, Hitool will display the first pair of non-matching addresses.

-B specifies compare the memory in Byte format.

-Hw specifies compare the memory in Half Word format.

-W specifies compare the memory in Word format.

It defaultly uses the current memory access mode which specified by the Memory Access Size command in Option menu.

Keyboard Examples:

COMpare 10 1F 20

Copy

Copy adr1 {adr2 | Length length} adr3 [-B | -Hw | -W]

adr1	specifies the first location in the memory range to be copied.
adr2	is a hexadecimal address specifying the last location in the memory to be copied.
Length	is the required prefix when specifying a "length" value.
length	is a decimal value specifying the memory size to be copied. This value is from 0x1 to 0x100000000.
adr3	is the starting address of the memory range where the specified source memory range is to be written.
-B	specifies copy the memory in Byte format.
-Hw	specifies copy the memory in Half Word format.
-W	specifies copy the memory in Word format.

The contents of the memory ranges beginning at adr1 are copied byte by byte to the destination range beginning at adr3.

It defaultly uses the current memory access mode which specified by the Memory Access Size command in Option menu.

Keyboard Examples:

Copy 0 1F 20	Copy memory 0 through 1F to 20 through 2F.

CReate

CReate #symbol adr

#symbol	is the symbol to be created with any (except “ #”) of the alphanumeric characters. "#" is used to prefix the created symbol.
adr	is the symbol address.

Any symbol created by this command is temporary and effective only during the current session. It will not be written back to the symbol file but may be saved as a disk file using the SSave command.

To delete an existing symbol, use Delete command.

Keyboard Examples:

CReate #AAA 5

DElete

DElete #symbol | $variable | All

#symbol	specifies the user defined symbol to be deleted. "#" must prefix the symbol to be deleted.
$variable	specifies the user defined variable to be deleted.
All	specifies all the user defined symbols and variables to be deleted.

This command is to delete an existing user-defined symbol or variable. Only symbols created by Create command or variables created by $variable command can be deleted with this command.

Keyboard Examples:

DElete #AAA

Disassemble

Disassemble [adr1 [adr2 | Length length]]

adr1	specifies the first location in the memory range to be disassembled.
adr2	is a hexadecimal address specifying the last location in the memory to be disassembled.
Length	is the required prefix when specifying a "length" value.
length	is a hexadecimal value specifying the memory size to be disassembled.

If no parameter is provided, disassembly will execute beginning at the current PC.

Keyboard Examples:

Disassemble 400	Disassemble program from address 400 to 40F

Fill

Fill adr1 {adr2 | Length length} {“string” | data..} [-B | -Hw | -W]

adr1	specifies the location in the memory where the fill operation is to begin.
adr2	is the location in the memory where fill is to end.
Length	is the required prefix when specifying a "length" value.
length	is an unsigned value (0x1 to 0x100000000) specifying the memory size to be filed.
string	is character string enclosed in quotation marks(“ ”). A string may not be more than 4 bytes in length. Up to 8 datum are accepted and a total of 32 bytes may be entered. Combined use of hex and string is permitted.
data..	is the data which to be written into memory.
-B	specifies fill the memory in Byte format.
-Hw	specifies fill the memory in Half Word format.
-W	specifies fill the memory in Word format.

To fill a memory range with a particular value, input the start address, the end address or length, and the desired value or string (in Byte, Half Word and Word format).

It defaultly uses the current memory access mode which specified by the Memory Access Size command in Option menu.

Keyboard Examples:

Fill 400 500 0x90 -B	Write the value of 90 to memory 400 – 500 in byte format

Go

Go [[Run] [From adr1] | [From adr1] [Till adr2] [Monitor timer]]

Run	Specifies free run. During free run all breakpoint settings are ignored, but target information is still recorded in the trace buffer.
From adr1	is the address where emulation is to begin.
Till adr2	Specifies a temporary breakpoint in the emulation. This breakpoint (adr2) exists only in this emulation session.
Monitor	Specifies to refresh the content of the opened window after a defined period. (like source, Register window etc.)
timer	Is the value of refresh window timer unit. It is limited from 1 to ff.

When you execute the Go command and there is no breakpoint in this program, the emulation CPU will start at the current program counter and go without stop. And if you want to stop the CPU you can use the “ESC” key.

Keyboard Examples:

Go	Execute emulation from current address

HAlt

HAlt

This command may be used during free run emulation to stop execution at any time.

Help

Help [command]

command	is the shell command you want to view its keyword and syntax. If no parameter is entered, an alphabetical listing of all available shell commands will display. Pressing <F1> will also provide the list.

Keyboard Examples:

Help

Include

Include filename

filename is the command file to be executed. It is a text file you can create or edit with a DOS based editor. You may also use the Log command to create a command file.

Keyboard Examples:

Create the include file TEST1.INC in the directory c:\temp with any edit tools as follows:

Memory 400 430

Fill 400 430 0x90 0x78

Include TEST1 Execute command file, filename: TEST1

Jump

Jump adr

adr is the command to change the program counter (PC) to hexadecimal address.

Change the emulation processor's PC counter to the specified address without perform emulation.

Keyboard Examples:

Jump 500 force PC=500

LOad

Load filename [Symbols | NOSymbols] [Code | NOCode] [Warn | NOWarn] [S-R | BIN addr | DIS addr | DISA | DISC addr | DISAC | EXEC]

filename specify the filename you want to load.

Symbols directs the loader to load symbols, defaultly set.

NoSymbols	skips loading symbols
Code	directs the loader to load binary code, defaultly set.
NOCode	skips loading code
Warn	displays warnings from the loader, defaultly set.
NOWarn	Supresses all warnings.
S-R	directs the loader to load code in S-Record format.
BIN	directs the loader to load code in binary format.
DIS	directs the loader to load code in disassembly format not including address and opcode section.
DISA	directs the loader to load code in disassembly format including address section.
DISC	directs the loader to load code in disassembly format including opcode section.
DISAC	directs the loader to load code in disassembly format including both address and opcode section.
EXEC	directs the loader to load the executable file.
addr	specifies the start address where the loader is to load code.

[Symbols | NOSymbols] [Code | NOCode] [Warn | NOWarn] is meaningless to [S-R | BIN | DIS*]. The default is EXEC or S-R format. Loader will auto-determine the format according to the file.

Keyboard Examples:

1. LOAd prime.elf	Load the executable file.
2. LOAd prime.elf DISC 0x10000	Load the executable file to the memory block starting from 0x10000 in disassembly format including opcode section.

LOG

LOG {filename [Command] [Append]} | {ON | Off | CLear}

filename is the file to store the logged shell commands.

Command is the command key word to log. The command may then be executed using the Include command.

Append is the option to append the newly logged information into an existing file. Otherwise existing data in the file is automatically overwritten.

On|OFf is the option to enable/disable logging of executed commands.

CLear is the option to clear the logged commands in log file.

Keyboard Examples:

LOG test1 Log key-in shell commands to file TEST1.

LONg

LONg adr data 1 [..data 8]

adr specifies the location where the Memory Modify operation is to begin of the emulation CPU.

data1..data8 is a hexadecimal value up to 4 bytes (0~FFFFFFFFH).

Note that data format in Long implies double words.

Keyboard Examples:

LONg 400 1234 8888 Writes double-word data directly.

Memory

Memory adr1 [adr2 | Length length] [-B | -Hw | -W]

adr1 specifies the memory location where memory display operation is to begin. It is a

	hexadecimal address representing a hex address in the memory space of the emulation CPU.
adr2	is the hexadecimal address specifying the last memory location where memory display ends.
Length	is the required prefix when specifying a "length" value.
length	is a hexadecimal value specifying the memory size to be dumped/displayed.
-B	specifies display the memory in Byte format.
-Hw	specifies display the memory in Half Word format.
-W	specifies display the memory in Word format.

To display the contents of memory in a specific range, input a starting address and either an end address or the desired length. If no end address is given, the contents of memory will be displayed one page at a time.

When an end address is specified the contents of memory are displayed continuously until the last location in the range is reached.

Memory contents are displayed in hexadecimal.

Keyboard Examples:

Memory 400	Display memory from 400.

QUEry

QUEry #symbol | hex_adr

#symbol	is the symbol name or line number of the address in query.
hex_adr	is the hexadecimal address setting of the symbol name in query.

To query for the address of a specified symbol or the symbol of a specified address. Hitool will search through the symbol table and list the specified symbol or address.

Keyboard Examples:

QUEry 60	Query the symbol corresponding to the address 60.

Register

Register [{reg_id} [data]]

reg_id is a general register related to the processor.

data is the new data to be assigned to the designated register.

This command displays or modifies the contents of the emulation processor registers. If "Register" alone (without any parameter) is entered, the contents of all of the emulation processor registers are displayed.

Keyboard Examples:

Register SP 0x8 Modify register SP to 8 directly.

RESet

RESet [adr]

adr is the address where PC is to set directly after reset. It is an unsigned value.

Use this Output command to reset/initialize the emulation process, It will perform a Reset action first, then set the PC to the specified address directly. Default address is zero or the entry point if the elf file loaded.

Keyboard Examples:

RESet Reset emulation processor.

RESTart

RESTart

Similar to Reset, but it would go from the start address to the C main entry if it is specified. You can specify the related option in Debug tab of Options dialog box.

SAvedisasm

SAvedisasm adr1 {adr2 | Length length} file

adr1 indicates the starting address of disassemble instruction to be saved.

adr2 indicates the end address of disassemble instruction to be saved. It may be prefixed.

Length is the required prefix when specifying a "length" value.

file is the name of file where disassembly instructions save into.

SAvedisasm is used for saving disassemble instruction within the specified address range into a file. Address must be hexadecimal values.

Keyboard Examples:

SAvedisasm 0 F C:\a Save disassembly instruction into file "a" from address 0 to F

SRecall

SRecall filename

filename is the symbol file where the to be recalled symbols are located.

Use this shell command to retrieve stored symbols from file. Symbols were previously created with Create command and saved with the SSave command.

This command is useful when retrieving stored symbols to insert them into a newly created symbol table. If symbols already exist in the new symbol file when the stored symbols are retrieved, the symbols from disk file are appended into the newly created symbol table.

Keyboard Examples:

SRecall TEST1.SYM

SSave

SSave filename

filename the filename to save and store the created symbols. If filename already exist, it will be overwritten.

This command is use to save the symbols created by Create command.

Keyboard Examples:

SSave TEST1.SYM

Step

Step [Over][count | FOrever]

Over is the option to treat subroutines(Call, INT) and loop instructions(REP, LOOP) as single step instruction. With this option, you need not stop into subroutines you are not concerned about. Hitool runs through such subroutines without displaying.

count is a hexadecimal value specifying the number of instructions to step.

FOrever is the option to step instruction continuously until forced to stop by pressing <Esc>.

After each step, the window information is also updated accordingly if the respective window updating toggle is enabled. During Step execution, NMI is disabled and other interrupts are masked.

Keyboard Examples:

Step Execute one Instruction

SYmbol

SYmbol [Global | Module | #module | Lines]

global is the option to display global information.

Module is the option to display symbol only from a module.

#module	is the option to display symbol from a specific module name.
Lines	is the option to display line address information of all modules. It includes the start address of a module and the start/end address of each module line.

This shell command is use to display symbols. If no parameter is given, all existing symbols are displayed.

Keyboard Examples:

SYmbol Global

TEst

TEst addr1 {adr2 | Length length} [Mem | Data | Addr] [-B | -Hw | -W]

adr1	indicates the starting point of the memory block to be tested.
adr2	is an unsigned value specifying the last memory location to be tested.
Length	is the required prefix when specifying a "length" value.
length	is a decimal/hexadecimal value specifying the memory size to be displayed. The default is decimal.
Mem	is to test memory.
DAta	is to test data bus.
ADdr	is to test address bus.
-B	specifies to test the memory in Byte format.
-Hw	specifies to test the memory in Half Word format.
-W	specifies to test the memory in Word format.

Memory test command is used to verify service ability of the memory, data bus and address bus. The test changes the original contents of the specified range.

It defaultly uses the current memory access mode which specified by the Memory Access Size command in Option menu.

Keyboard Examples:

Test 0 FF Test memory from 0 to FF.

Upload

Upload adr1 { adr2 | Length length } filename [S-R | BIN | DIS | DISA | DISC | DISAC]

adr1	indicates the starting point of the memory block to be uploaded.
adr2	is an unsigned value specifying the last memory location to be uploaded.
Length	is the required prefix when specifying a "length" value.
length	is a hexadecimal value specifying the memory size to be uploaded.
filename	is the file (with path) to store the uploaded data.
S-R	is to store the uploaded data in S-Record format. It is default.
BIN	is to store the uploaded data in binary format.
DIS	is to store the uploaded data in disassembly format not including address and opcode section.
DISA	is to store the uploaded data in disassembly format including address section.
DISC	is to store the uploaded data in disassembly format including opcode section.
DISAC	is to store the uploaded data in disassembly format including both address and opcode section.

This shell command is used for uploading memory code by formatted hex record to a file.

Keyboard Examples:

Upload 0 FF test1 BIN

WAtch

WAtch "C-Expression"

"C-Expression" specifies variable or c expression. The quotation mark (" ") must be entered.

Keyboard Examples:

WAtch "$a++"

Watchpoint

Watchpoint [ADdr addr] [AMask mask] [DAta data] [DMask mask] [-B | -W] [Fetch | Access] [Read | Write] [HW0 | HW1]

HW0 specifies HW0 Watchpoint.

HW1 specifies HW1 Watchpoint.

ADdr is the prefix when specifying the address of Watchpoint

addr specifies the address of Watchpoint.

AMask is the prefix when specifying the address mask of Watchpoint

mask specifies the address mask of Watchpoint.

DAta is the prefix when specifying the data of Watchpoint

data specifies the data of Watchpoint.

DMask is the prefix when specifying the data mask of Watchpoint

mask specifies the data mask of Watchpoint.

-B specifies access memory in Byte format.

-Hw specifies access memory in Half Word format.

-W specifies access memory in Word format.

Fetch specifies Instruction Fetch.

Access specifies Data Access.

Read specifies only after meeting Read Data to memory condition will watchpoint function.

Write	specifies only after meeting Write Data to memory will watchpoint function.

For more detailed information about Watchpoint, please refer to Chapter 7.

Keyboard Examples:

Watchpoint1 ADdr 34 –B Fetch	Set Watchpoint1 at memory 34

NOTE

For Angel series, Watchpoint command only supports Write, Access, Address, -B|-Hw|-W. –B means the watchpoint is ET type, while –Hw and –W mean the watchpoint is BITMASK type.

Word

Word adr data1 [.. data16]

adr	specifies the location where the Memory Modify operation is to begin. It is a hexadecimal address representing a hex address in the space of the emulation CPU.
data1..data16	is a hexadecimal value of up to 2 bytes (0~FFFFH).

Note that data format in Word, implies words.

Keyboard Examples:

Word 400 0xA3	Write 0xA3 to memory 400 405

13 Semihosting And Misc

Overview

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism could be used, for example, to allow functions in the C library, such as printf() and scanf(), to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

Semihosting is implemented by a set of defined software interrupt (SWI) operations. The application invokes the appropriate SWI and the debug agent then handles the SWI exception. The debug agent provides the required communication with the host. In many cases, the semihosting SWI will be invoked by code within library functions. The application can also invoke the semihosting SWI directly.

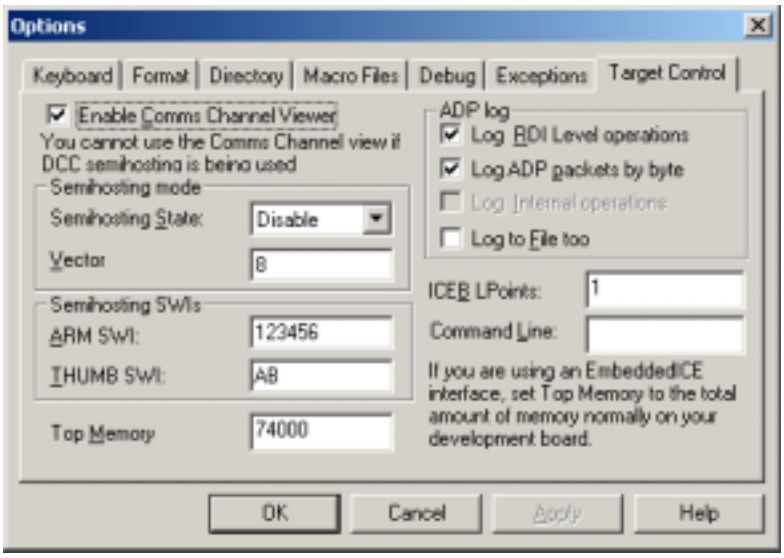
NOTE

Semihosting is not available for GCC build environment.

Semihosting Handle

Hitool for ARM handles the semihosting in similar manner for vary protocol. For Angel and Jeeni protocol, Angel debug monitor or Jeeni converter monitors the semihosting, but for the other protocol, the host debugger monitors the semihosting. Some semihosting functions are implemented by the debugger.

You can set some setting to control the manner in Target Control tab of Option dialog box as following:



NOTE

If the target does not support semihosting or each of semihostings, the corresponding item will be grayed.

Usually users had better not change the semihosting setting unless users change the default SWI instruction setting during application.

In addition to using semihosted SWIs, many applications will need to install their own SWI handlers into the vector table. Then the semihosting vector must be modified to point to a location at the end of the application handler that is only reached if your handler does not recognize the SWI (or recognizes it as a semihosting SWI).

Note that the instruction at the vector address never gets executed because the debugger returns directly to the application after processing the semihosting SWI. However, using a normal SWI return instruction ensures that the application does not crash.

The ARM C library startup routines use system SWIs to set up the heap and stack. In this case, the vector must initially point to the system vector until the application is ready to install its handler. So you should set a breakpoint in the application code just before the application handler is installed to change the vector.

In Target Control tab, you also can set ADP log, Command Line, Semihosting, argument etc.

NOTE

ADP log group is only available for Angel or Jeeni protocol.

Command Line option specifies the argument of the image debugging.

ICEBreaker lockpoint shows or sets the locked EmbeddedICE logic units.
If "Log to File too" option is checked, the device driver level data or information is logged into a file named Rdi.log. This option is based on the device level.
If "Log ADP packets by byte" option is checked, the device driver level data is logged. ">" prefix is to write data, "<" prefix is to read data.
If "Log RDI Level operations" option is checked, RDI level information is logged, such as "entered RDI info", "waiting for message" and so on. And then RDILog pane will appear in Output Window. In addition, Target Information is only available for Angel and Jeeni protocol.

Top Memory indicating the max memory address that can be used by Semihosting. Make sure the Top Memory is available for the target board, which indicates the RAM.

RDILog pane

RDILog Pane displays the low-level communication messages between Hitool for ARM and the target processor.

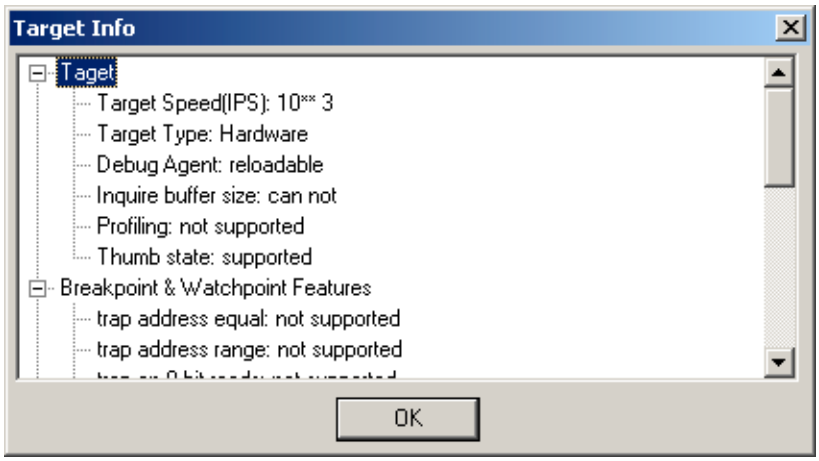
DEBUG: ADP_ICEB_Exists.
>1C >00 >00 >18 >01 >5F >5E >01 >1B >51 >00 >01 >0
>00 >FF >FF >FF >FF >FF >FF >FF >00 >00 >01 >0
>1B >5B >1D
wait_for_debug_message waiting for 80010011
<1C <00 <00 <18 <01 <5F <5F <01 <00 <00 <01 <80 <0
<FF <FF <FF <FF <FF <FF <FF <FF <1B <51 <00 <01 <0
<AA <1D
wait_for_debug_message got packet

RDI interface does not implement stepping function. So during stepping an instruction, the debugger will analyze the code first, then insert a breakpoint, go, halt, then remove the breakpoint. Whether

RDI support instruction stepping is displayed in Target Info dialog box.

Target Info dialog box

Simply click on Target Info in Help menu, Target Info dialog box will pop-up as the following:



It displays some information about RDI interface and Angel debug monitor. You can expand the items to get more information.

Comms Channel Viewer

NOTE

The Comms Channel Viewer is only available for the Embedded ICE logic.

The Embedded ICE logic in ARM cores such as ARM7TDMI and ARM9TDMI contains a debug communications channel, which enables data to be passed between the target and the host debugger through JTAG port without stopping the program flow or entering debug state.

The DCC sample is available in Hitool for ARM installation directory /examples/ARM/DCC, For more information about how to build and execute these examples, please refer to the Readme.txt file in the same directory.

By the way, the DCC semihosting can not be implemented so far.

You can enable or disable the Comms Channel Viewer in the Output Window by checking or unchecking the Enable Comms Channel Viewer Option in the target Control tab if available.

NOTE

Comms Channel Viewer is only available for Jeeni now.

Comms Channel viewer allows you to examine data passed to and passing from the debugger target through the Debug Communications Channel (DCC) and to send data of your own.

Comms Channel viewer consists of two areas. They are Send area and Received area. Each of them has different function.

Send area of this viewer is to send information down the channel.

To send information, type information in the edit box then click on the Send button on the right. If you want to input data from a file, select the Read from file item from the pop-up menu.

The information will be word-aligned and stored into the buffer automatically. The information will be sent in ASCII character code when requested by the target.

Receive area is to send counter displaying the number of bytes left in the buffer. The information received is displayed in ASCII character code in Receive area.

To log the received information into a file, select the Log to file item from the pop-up menu.

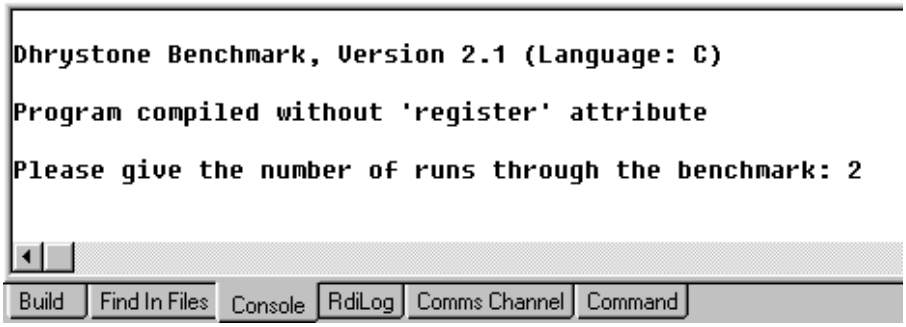
Console view

The Console pane allows you to enter data from your keyboard to the program being debugged, when it might normally receive data from some other device, and to display on your screen output that might normally be sent elsewhere.

You might want to debug an image that is intended to receive input from or write output to devices that are not yet available. The Console view provides the semihosting facility that allows you to do so.

Output from an executing image is displayed, and you can respond by entering data from your keyboard or from a file to provide input for the image.

A typical Console pane view is shown as below:


Dhrystone Benchmark, Version 2.1 (Language: C)
Program compiled without 'register' attribute
Please give the number of runs through the benchmark: 2
Build
Find In Files
Console
RdiLog
Comms Channel
Command

A Compiler Configuration and JTAG Port

Compiler Configuration

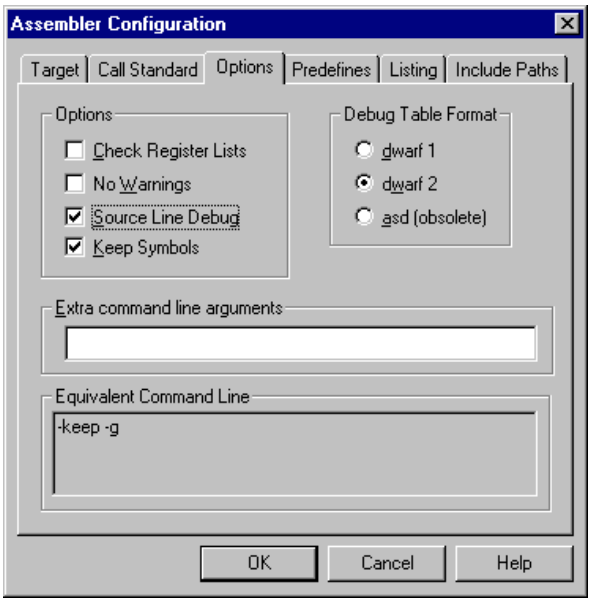
Here we introduce some important options you should pay attention to if you want to debug target file built under

- ARM compiler environment
- ADS compiler environment

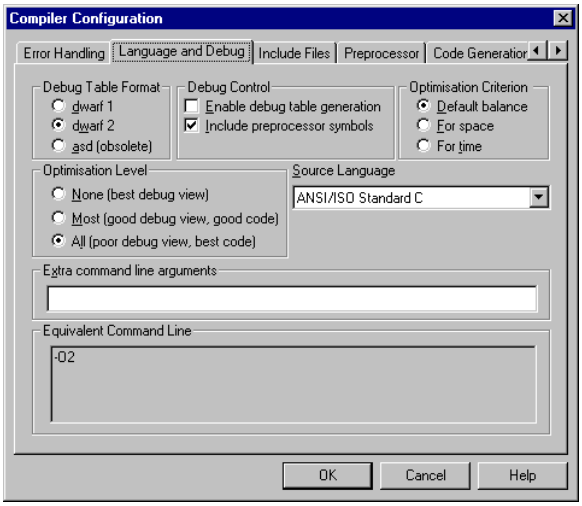
ARM Compiler Configuration

Assembler and Compiler Configuration

In the following Assembler Configuration dialog box Options tab and Compiler Configuration dialog box Language and Debug tab, you should choose dwarf format in Debug Table Format section for Hitool does not support asd debug format.

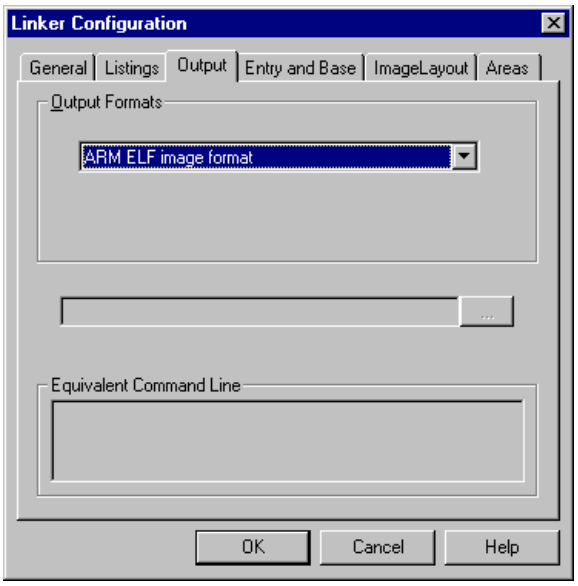


To fully debug, you had better choose Source Line Debug and Keep Symbols options in Assembler Configuration dialog box and Optimization Level NONE option in Compiler Configuration dialog box.



Linker Configuration

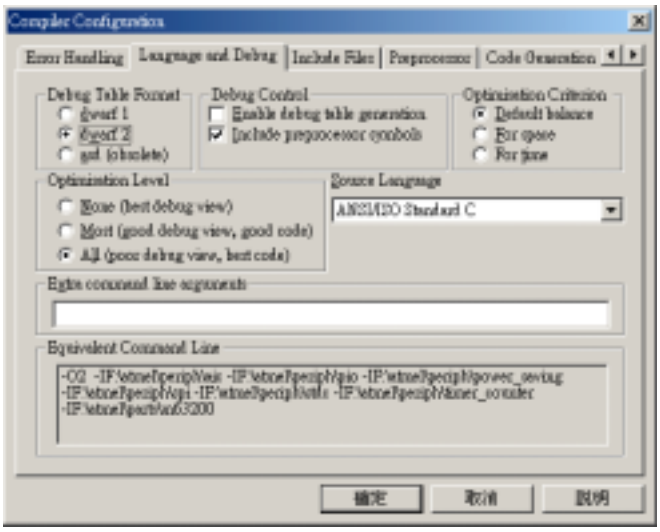
In the following Linker Configuration dialog box Output tab, you should choose ARM ELF image format for Hitool only supports ARM ELF format.



ADS Compiler Configuration

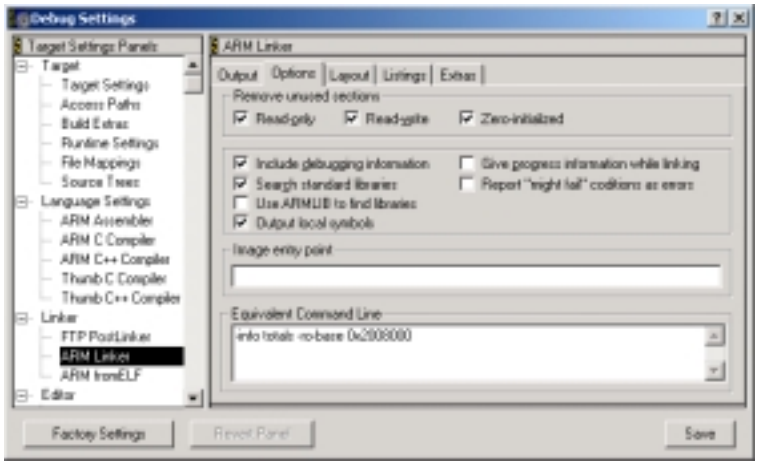
ARM C Compiler Configuration

In the following ARM C Compiler panel Debug/Opt tab, you should choose Enable debug table generation in Debug Control section for Hitool needs debugging information during debugging.

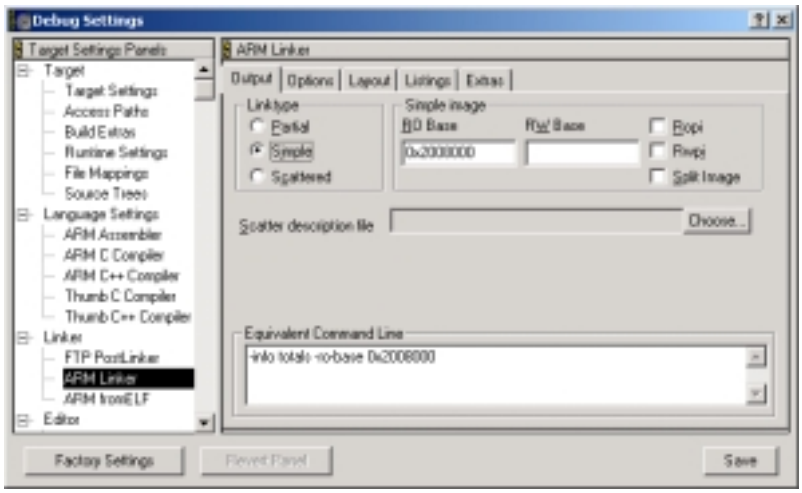


ARM Linker Configuration

In the following ARM Linker panel Options tab, you should choose Include debugging information to provide debugging information during debugging.



You can set RO Base and RW Base addresses separately in the RO Base and RW Base fields in the following Output tab of ARM Linker panel according to the evaluation board you are using.



JTAG port

The 20-pin connector (J1) is connected to the JTAG interface of the microcontroller. The pinout is compatible with the ARM Multi-ICE interface unit. A pinout of the JTAG connector is shown in the following Figure.

Vcc	**1**	**2**		**Vcc**
NTRST	**3**	**4**		**GND**
TDI	**5**	**6**		**GND**
TMS	**7**	**8**		**GND**
TCK	**9**	**10**		**GND**
NC	**11**	**12**		**GND**
TDO	**13**	**14**		**GND**
NRESET	**15**	**16**		**GND**
NC	**17**	**18**		**GND**
NC	**19**	**20**		**GND**

Pinout of JTAG connector (J1)

Vcc Power supply from ARM7TDMI evalution board.

NRESET This signal resets the ARM7TDMI processor core.

NTRST This signal resets the tap controller and EmbeddedICE logic. This resets the internal functionality used by any debugger or other device connected to the JTAG port.

TDI JTAG Test Data In. The TDI level is used to serially shift test data and instructions into the ARM7TDMI during JTAG test operations.

TMS JTAG Test Mode Select. This pin controls JTAG test operations in the ARM7TDMI.

TCK JTAG Test Clock. The JTAG test clock shifts state information and test data into, and out of, ARM7TDMI during JTAG test operations.

TDO JTAG Test Data Out. The TDO level is used to serially shift test data and instructions out of the ARM7TDMI during JTAG test operations.